

# SFANDROID

Developed by

YAMEL SENIH

CARLOS PARADA

Migrated by

REDHUAN D. OON

Version 1.8b



## RED1's THREE LAWS:

**Information is Free**

YOU HAVE TO KNOW

**People are Not**

YOU HAVE TO PAY

**Contributors are Priceless**

YOU HAVE TO BE

---

# Sales-Force Android

## Concept Mobile App for iDempiere

A ground breaking killer ERP solution replicated to a full Android application.

Upgraded to latest iDempiere 1.0c, reviewed and documented for community technical improvements.



**June 2, 2013 Canoa Quebrada, BRAZIL** - I was next to go flying down into this desert dune lake created from millions of years of evolution on the north-east beach front facing the Atlantic Ocean.

## TABLE OF CONTENTS

|                                               |    |
|-----------------------------------------------|----|
| Disclaimer and Credits                        | 6  |
| On The Go                                     | 7  |
| <i>Album</i>                                  | 8  |
| <i>Progress Worksheet</i>                     | 9  |
| <i>Note on Version 1.6</i>                    | 10 |
| <i>Note on Version 1.7</i>                    | 11 |
| <i>Note on Version 1.8</i>                    | 11 |
| <i>Note on future versions</i>                | 11 |
| Overview                                      | 12 |
| <i>Introduction</i>                           | 12 |
| <i>Application Architecture</i>               | 13 |
| <i>Glossary</i>                               | 14 |
| Menu Items                                    | 15 |
| <i>Mobile System Menu</i>                     | 15 |
| <i>Initial Load Window</i>                    | 17 |
| <i>Initial Load - Sequence</i>                | 18 |
| <i>System Sequence</i>                        | 18 |
| <i>Menu Mobile System</i>                     | 19 |
| <i>Mobile Other Action</i>                    | 21 |
| <i>Menu Mobile Synchronization</i>            | 21 |
| <i>Generate Web Service</i>                   | 22 |
| <i>Web Service Security Window</i>            | 22 |
| <i>Generate Sequence Mobile</i>               | 23 |
| Sales Force Module                            | 24 |
| <i>Concept of Verticals</i>                   | 24 |
| <i>SalesForce Menu</i>                        | 24 |
| Setting Up                                    | 26 |
| <i>Direct Hot Setup</i>                       | 26 |
| <i>Downloading the SFAndroidServer Plugin</i> | 26 |
| <i>At the OSGi Console</i>                    | 27 |
| <i>Sequence Check</i>                         | 28 |

|                                       |           |
|---------------------------------------|-----------|
| <i>Mobile Data Pack-In</i>            | 29        |
| <i>Role Access Update</i>             | 31        |
| <b>Developer's Note</b>               | <b>32</b> |
| <i>BitBucket Source</i>               | 32        |
| <i>SourceForge.net Repository</i>     | 32        |
| <i>2Pack Exporting</i>                | 33        |
| <i>GardenWorld Sample</i>             | 34        |
| <i>SFAndroid Web-Services</i>         | 36        |
| <i>POWrapper</i>                      | 37        |
| <b>Android Development</b>            | <b>38</b> |
| <i>Using the ADK</i>                  | 38        |
| <i>Launching via Emulator</i>         | 40        |
| <i>Login Code</i>                     | 43        |
| <i>SOAP Request</i>                   | 45        |
| <i>SOAP Response</i>                  | 48        |
| <i>Synchronizing Data</i>             | 50        |
| <i>Trouble-shooting Synch Scripts</i> | 52        |
| <i>Using ADB command line tools</i>   | 53        |
| <i>Wipe-out User Data</i>             | 54        |
| <b>Embedded Demo</b>                  | <b>55</b> |
| <i>Debug ADK on real Phone</i>        | 56        |
| <i>Source of Embedded Sample</i>      | 58        |
| <i>SQLite Data Admin</i>              | 59        |
| <i>SFAndroid Meta Menu</i>            | 60        |
| <i>Your chance to contribute</i>      | 62        |

## DISCLAIMER AND CREDITS

THIS IS AN OPEN SOURCE AND LIBRE SOFTWARE PROJECT WITH NO WARRANTIES WHATSOEVER.  
THIS IS A COPYLEFT DOCUMENT INTENDED FOR FREE DISTRIBUTION AT NO COST:

<http://sourceforge.net/projects/redt/files/ADempiere%20PDFs/SFAndroid.pdf/download>

NO PART OF THIS DOCUMENT CAN BE REFERRED OR TRANSLATED WITHOUT THIS ENTIRE PAGE INTACT WITHIN THE NEW DOCUMENT. ANY DERIVATIVE WORK THAT CONTAINS THIS DOCUMENT MAY BE SOLD AT A PRICE WITHOUT ANY ROYALTY TO THE AUTHOR(S). HOWEVER A SOFT COPY MUST BE MADE AVAILABLE WITHOUT COST. THE LINK SHOULD BE SENT TO THE AUTHOR FOR REFERENCE.

TRANSLATIONS ARE ENCOURAGED AND APPRECIATED BY THE GLOBAL FREE ERP COMMUNITY.

DOCUMENT - SALES FORCE ANDROID - THIS VERSION IS 1.8, DATED September 12, 2013 - ongoing.

APPLICATION DEVELOPERS - YAMEL SENIH / CARLOS PARADA

[ysenih@erpconsultoresyasociados.com](mailto:ysenih@erpconsultoresyasociados.com) / [cparada@erpconsultoresyasociados.com](mailto:cparada@erpconsultoresyasociados.com)

PROJECT MANAGER - ANGEL MUNOZ [angelfmr@erpconsultoresyasociados.com](mailto:angelfmr@erpconsultoresyasociados.com)

**ERP CONSULTORES Y ASOCIADOS - VENEZUELA**

PROJECT PARTNER - JOSE BOTERO (**LUBO SISTEMAS - VENEZUELA**) [jose.botero@lubosistemas.com.ve](mailto:jose.botero@lubosistemas.com.ve)

CLIENT USER MANAGER - MIGUEL RONDON (**ACROSILOS - PANAMA**) [glapmigue@gmail.com](mailto:glapmigue@gmail.com)

SPECIAL THANKS TO MY GENEROUS HOSTS FOR MY PASSAGE AND WELL-BEING THROUGHOUT -  
ANTHONY SOOSAH - ELECTRIC COMPANY LTD, GHANA (PASSAGE TO GERMANY)

THOMAS AND MICHAELA BAYEN - WITHIN GERMANY

WERYSSON DANTAS AND EDILSON NETO - WITHIN BRAZIL AND PASSAGE TO VENEZUELA

ORLANDO CURIELESS/MARIANA, and MIGUEL HERNÁNDEZ - WITHIN VENEZUELA

PLUGIN UPGRADE & AUTHOR OF THIS DOCUMENTATION - REDHUAN D. OON [redi@redt.org](mailto:redi@redt.org)

SPONSOR - ZEESHAN (SYSNOVA - BANGLADESH) [zeeshan@gmail.com](mailto:zeeshan@gmail.com)

### LATER CREDITS

DEEPAK PANSHERIYA, for assisting in web-services trouble-shooting and solving it in new plugin deployment.

MIGUEL HERNANDEZ, for translating to Spanish language.

## ON THE GO

When I finished the last mission which is the iUI Mobile conversion into an OSGi plugin for iDempiere, I was on the way back from Siberia to Germany to participate in the first iDempiere training and also the first iDempiere world conference in Krefeld.

At the same time, one of the attendees, Edilson Neto (pronounced ‘Air-zeal-sen’) from Brazil negotiated a further travel arrangement for me to be in Latin America to coach and assist his team in Fortaleza, Brazil. I offered to do that provided I am given combined passage to Venezuela, a country I have not set foot on before this.

Another reason was due to my next mission to convert the SF Android application contributed by a Venezuelan team but its code was undergoing revision and not well documented. Thus been in Venezuela will allow me to seek out that team. Fortunately the Venezuelan ADempiere/iDempiere contributor company DCS under Orlando Curieless established communication with the SFAndroid group and have them visited us in Barquisemeto when I was there.

In a full day of interacting from morning coffee with lunch in between, with constant translation as I do not know Spanish and the lead developers do not know English, we began to build a starting point of understanding on how to collaborate on the project. Needless to say there were a lot of *oohs*, and *aahs*. Also *uh-uhs*.

After leaving Venezuela before end of June, and now here in Japan, attending the ADempiere Japan leader’s wedding and meeting with the giant Nomura Research Institute, I continue the constant and consistent remote *Google-Translator* assisted online communication with the lead developer, Yamel Senih to fully convey the concept and documentation needs in order that I can fully complete the upgrade of the plugin for iDempiere usage with the usual complete tutorial guide.

This guide is the first version with a later version describing the Android mobile-side application.

Redhuan D. Oon  
July 8th, 2013,  
Tokyo, JAPAN

## Album

As usual so as not to kill the sheer boredom that goes with writing a highly technical document, I have interjected it with some photo-log at strategic locations. I am sure it helps with the sheer boredom that goes with reading a highly technical document.



May 23rd, towards Czech border. The famous no-speed-limit-autobahn has to be experienced in Germany's own signature brand by Carlos Ruiz, Juan David Arbodela and me in the back seat. This was after the iDempiere Conference in Kre-

feld. Juan then took over and managed to floor past 235 kph before I stopped filming and jumped for cover. Yes I know. He is nuts. They then dropped me in Prague where I continued to Bratislava and Budapest with Norbert Bede, our Slovakian project member who introduced me to Pavol Kustar who became my personal video maker.



The photos are taken on the trail of my lifestyle of part-project, part-travel to meet community members, and part-taking pleasure in the leisure that one often associate with wayfaring. I do keep a more luxurious supply of candid shots in my online albums at Facebook, which has the potential of derailing this technical document. Thus I have made the appropriate selection to amuse in the minimalist way possible.

## Progress Worksheet

| Version    | Items                                                                                                                                    | Location            |
|------------|------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| 1.0        | Main document - SFAndroid-Server plugin                                                                                                  | Tokyo - July 8      |
| 1.5        | Web-service plugin                                                                                                                       | Langkawi - August 9 |
| 1.6        | Update with TableIndex, 2Pack changes                                                                                                    | Kuang, August 11    |
| 1.7        | Soap Request-Response illustration                                                                                                       | Kuang, August 28    |
| 1.8        | Binary WS solved. How to synchronize data. Debugging issues during InitialLoad. Embedded Demo on phone. SQLite admin. Examine Meta Menu. | Kuang, September 12 |
| - future - | - complete mobile side functionality/document -                                                                                          |                     |

| Action                                                            | Version-Date                                                                                                                                                    | Results                                                                              |
|-------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| Test on local instance                                            | <a href="#">Jenkins - August 13, 2013 Build #221</a><br><a href="#">SFAndroid-Server.jar</a> , <a href="#">MobileSynch.zip</a> , <a href="#">MobileData.zip</a> | Start 2Pack issue. Manual PackIn OK                                                  |
| - GardenWorld data                                                | - same -                                                                                                                                                        | PackIn of MobileData OK, all resolved.                                               |
| Using Felix console                                               | - same -                                                                                                                                                        | OK to install plugins                                                                |
| <a href="http://demo.idempiere.com">http://demo.idempiere.com</a> | August 13                                                                                                                                                       | Manual PackIn Failed - Validator error                                               |
| Test web service                                                  | <a href="#">SFAndroid-WS.jar</a>                                                                                                                                | Working in Eclipse IDE, awaiting core to export idempiere-webservice dependencies.   |
| Apply latest main code, scripts. Test everything.                 | <a href="#">Build as of #228 (Aug 27, 2013 5:03:35 AM)</a>                                                                                                      | Similar to above. Core WS updated. Deployed plugin no error but passive. Now solved. |

| Resource    | Link                                                                                                                            | Updated                                   |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| Source-code | <a href="http://bitbucket.org/red1/">http://bitbucket.org/red1/</a>                                                             | Some private, write to author for access. |
| Forum       | <a href="http://red1.org/adempiere/viewtopic.php?f=29&amp;t=1765">http://red1.org/adempiere/viewtopic.php?f=29&amp;t=1765</a>   | this version 1.8                          |
| Repository  | <a href="http://sourceforge.net/projects/red1/files/p2/SFAndroid/">http://sourceforge.net/projects/red1/files/p2/SFAndroid/</a> | this version 1.8                          |

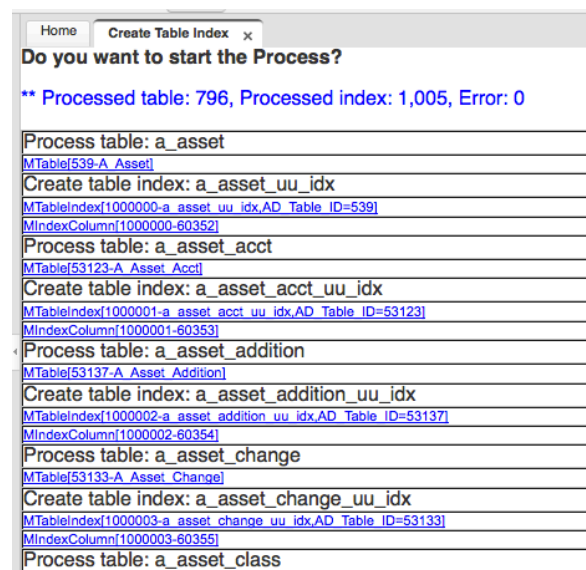
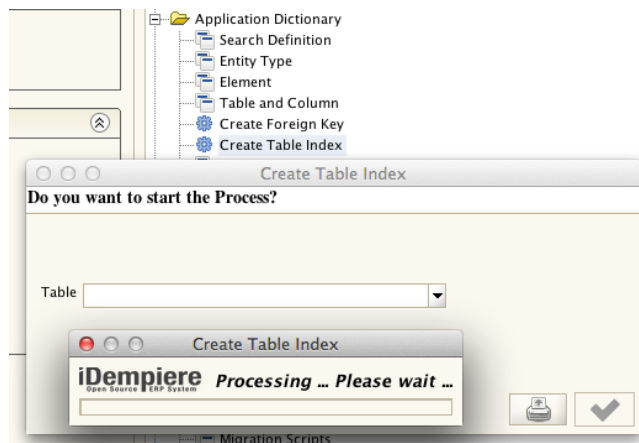
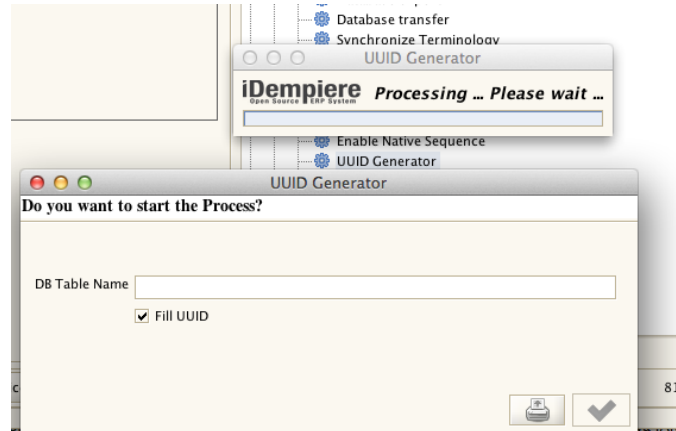
## Note on Version 1.6

Latest code changes require that I update with the new **FillUUID** during *UUID Generation*, and the all new **TableIndex** generation before redoing the 2Packs. This change potentially breaks all previous 2Packs, thus my upgrading this was vital.

The *Fill UUID* checkbox allows all legacy records to be filled up with the newly introduced UUID as backup to the PK regime that survived since Compiere days. UUID introduced by Heng Sin is critical to the multi tenant nature of the ERP to avoid conflict when more instances of the ERP are merged together particularly via 2Packs.

Another process done is the new **TableIndex** generation. You can read more of this new technical enhancement in the [IDEMPIERE-1132 ticket](#).

You need not do these two as I have done it as `ExpDat<timestamp>.jar`. Extract this with `jar -xvf` command to get the `ExpDat.dmp` for **RUN\_DBRestore** execution into your *PostgreSQL* database. Note that I do not use *Oracle* database version. But such a contribution following my steps here is welcomed.



## Note on Version 1.7

Deepak Pansheriya helped solve the WS calls from Mobile to the Server where Style.DOCUMENT is preferred.

Tracking of the SOAP communication and app handling to and fro, with final creation of SQLite database on the mobile phone to prove the concept works. Only InitialLoad is tested.

## Note on Version 1.8

Deepak Pansheriya helped solve the WS binary deployment. Below is the proof of it.

```
480    ACTIVE    org.idempiere.webservices_1.0.0.v20130827-0317
481    ACTIVE    org.restlet_2.1.3.v20130823-0839
482    RESOLVED   org.restlet.ext.servlet_2.1.3.v20130823-0839
483    ACTIVE    org.zkoss.zk.library_6.0.0.v20130827-0317
487    RESOLVED   org.idempiere.SFAndroidWS_1.0.0.201309100507

osgi> start 487
10:55:10.114 [/AppDroidServices].log: Initializing Spring root WebApplicationContext [29]
10:55:10.115 ContextLoader.initWebApplicationContext: Root WebApplicationContext: initialization start
10:55:10.116 XmlWebApplicationContext.prepareRefresh: Refreshing Root WebApplicationContext: startup d
55:10 MYT 2013]; root of context hierarchy [29]
10:55:10.116 XmlBeanDefinitionReader.loadBeanDefinitions: Loading XML bean definitions from class path
/cxf/services.xml [29]
10:55:12.013 DefaultListableBeanFactory.preInstantiateSingletons: Pre-instantiating singletons in org.
ns.factory.support.DefaultListableBeanFactory@65fe58e0: defining beans [AppDroidServices]; root of fac
10:55:12.022 ReflectionServiceFactoryBean.buildServiceFromClass: Creating Service {http://www.erpconsu
/}AppDroidServices from class org.mb.appdroid.AppDroidServices [29]
10:55:12.040 ServerImpl.initDestination: Setting the server's publish address to be /AppDroidServices
10:55:12.042 ContextLoader.initWebApplicationContext: Root WebApplicationContext: initialization compl
]
```

A section on Embedded Demo is added illustrating for the developer how it is handled in code and examined on the real phone. Troubleshooting tips and SQLite admin included. Tracing how the meta menu is created via the MenuList linking to Forms.

## Note on future versions

Future versions will trace the mobile app functions and how it is actually used in real business scenario.

## OVERVIEW

### Introduction

This is a ground-breaking first-time concept application killer-app written for the Free ERP environment based on iDempiere OSGi framework. The server-side is an independent plugin that setup automatically the mobile content meta-data without any compilation from the core iDempiere ERP application.

The Android application for the mobile phone can pick up the meta-data when communicating with the remote iDempiere's web-service plugin. It can then pull data organised into the phone and operate as a standalone. Later it can replicate new data both ways with the ERP back-end upon synchronising.

This application is written by Yamel Senih and Carlos Parada under a project managed by Angel Munoz of **ERP Consultores y Asociados** and Jose Botero of **Lubo Sistemas**, from Venezuela.

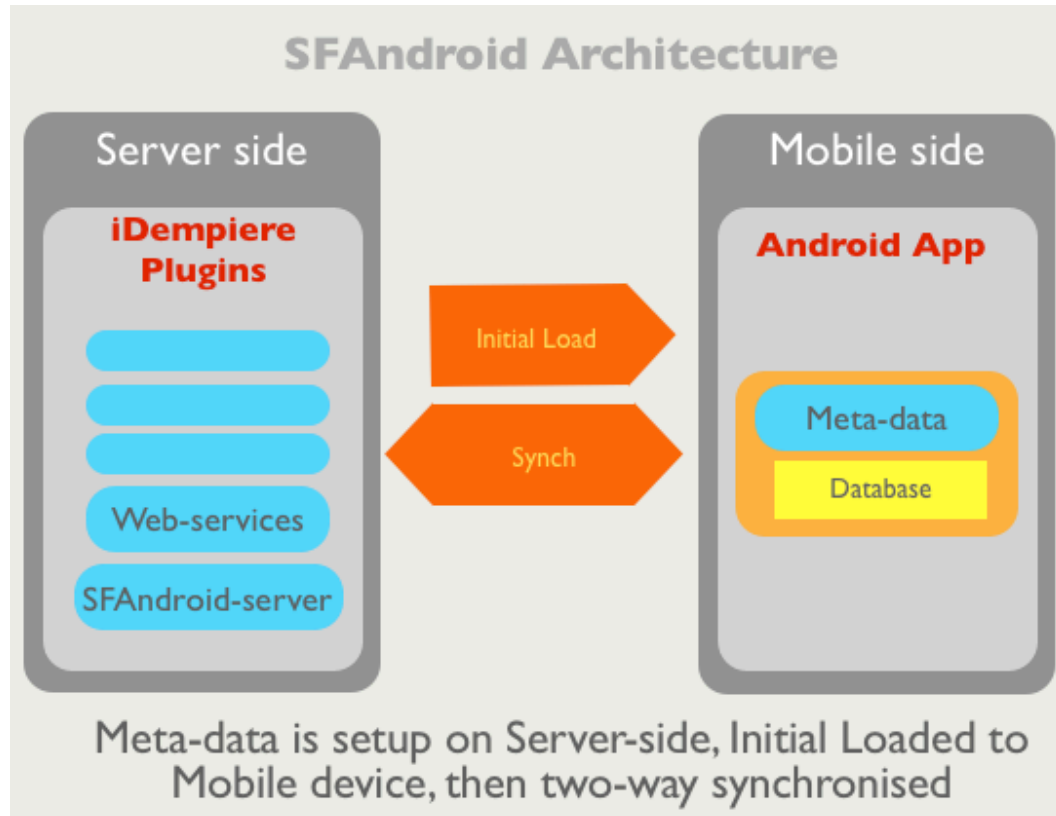
The mobile side was concept-proven in person to the author by its creators, but still under refactoring with the latest web-services from iDempiere and thus detailed illustration will be presented in the second version or section of this document due in some weeks time. Any mobile side description done here is only described conceptually. The focus of this earlier section is about the successful conversion of the server-side into an OSGi plugin within the iDempiere stack.



June 21 Barquisemeto - Team Venezuela: Angel Munoz, Jose Botero, Orlando Curieless, (me), Yamel Senih, Carlos Parada, Miguel Hernández.

## Application Architecture

We will examine in more graphic detail the layout of the application. The application is divided into two parts - the server side and the mobile client side. The illustration below acts as a starting and simple reference on the key relationship between the two sides.



Firstly the mobile side is also cleverly using the same meta-data concept used on the ERP side. For those who are familiar with Compiere's *Application Dictionary* or AD will know that the menus, and its windows, tabs and fields are defined by meta-data and not hard-coded. Similarly this Android application uses the same philosophy in defining its menu and windows.

However the creators of this app goes one step further by defining such meta-data on the server-side! It is then loaded onto the mobile client during running of the InitialLoad routine when the Android tries to synch with the web-service remotely. This concept is quite a novelty in itself. Let me explain further.

After the InitialLoad process, further synchronisation by the mobile phone will detect new data and replicate between the mobile device and the server. This is rather transparent to

the user. This idea also mean that there is centralised control of the mobile clients by the server-side.

The Android app is thus reduced to just the functional code but selectively used if allowed and described by the meta-data. Yes, even the code classes are defined by the meta-data on the server-side.

This means that the modification or updating of the code on the mobile side is minimised. Therefore the 'AD' approach allows the same flexibility to modify the menu format of mobile clients without updating the Android app.

I took the liberty to express my personal and pleasant surprise to the development team in Venezuela that this is truly marvelous ingenuity of the best a copycat can be. I also give them advanced praise that this may be a killer app that the world of ERP has been looking for. They even demonstrated to us how the Android can now do WorkFlow approval as required.

## Glossary

**AD - Application Dictionary**, is Compierre's architecture that was copied by its forks such as OpenBravo, ADempiere and iDempiere, where data defines its application structure of menus, windows, tabs, fields, data types and references with other abstract but common behaviour in a Financials ERP system.

**Mobile App** - the application software that you can install on your mobile phone that runs Android operating system or compatible with it. There are other interchangeable terms such as mobile phone app, Android app, mobile application and they all effectively means the same thing. Sometimes we can also call it by the project name - SFAndroid or refer to it as the client-side.

**Server-side ERP** - the iDempiere ERP system which oftens runs on a host server with access to its main database and connected to the Web in order for it to function fully. The term server-side is often emphasised to differentiate it from the client-side.

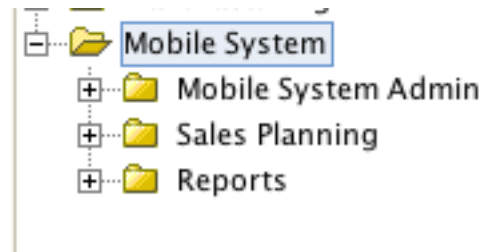
**WS** - Web Services. It is the web-based middle ware or interface by which other systems such as mobile apps can communicate with another system in this case the ERP server.

## MENU ITEMS

### Mobile System Menu

We take a first look at the new menu structure called Mobile System. (Go to the Systems Setup section at the bottom to install the plugin with 2Pack first).

Open up the iDempiere main menu screen and you can see the menu tree as shown below.



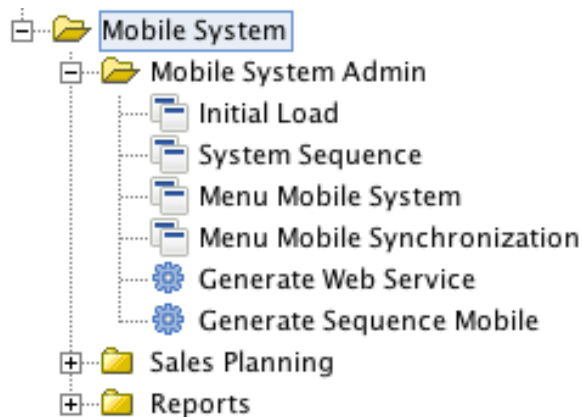
The Mobile System menu contains 3 sub-menus:

- I. **Mobile System Admin** - used to setup, define and configure the synchronising behaviour with the mobile application. The mobile menu and structure is also defined from here and at the server-side.
- II. **Sales Planning** - operates the *Sales Force* functionality for this mobile application.
- III. **Reports** - some reports used in *Sales Force* functionality.

What is important here is to regard item I as the main horizontal of the Android app integration and items II and III as the vertical SF (SalesForce module). The use of the items in Mobile System Admin is to manage the integration and communication between the ERP and the Mobile app, and thus can be re-used for any other vertical.

We go into the Mobile System Admin menu:

- I. **Initial Load** - Defines the metadata for uploading into the mobile application. 'AD' type meta-data is the concept used to first define the structure of data models, its tables and fields. Later at menu item IV, actual data can be trafficked between the ERP and the mobile app.
- II. **System Sequence** - determines the Document Sequence numbering format for the data in use by the mobile application. Later in VI it is generated based on this.
- III. **Menu Mobile System** - The menu structure and type of menu item that appears on the mobile phone application. This is the classic reuse of Compiere's Application Dictionary that allows flexibility in changing structure and behaviour without coding. It is defined on the server-side and sent to the mobile application during Initial Load above.
- IV. **Menu Mobile Synchronization** - This controls the actual data content of the system. Includes the traffic and direction of operational data when the synchronization icon is pressed on the mobile phone application. It also defines other properties of each item such as type of web-service used, image icon file reference, and *Rules* before and after the event. Both III and IV are of the same Table-Column model, differentiated by its *Menu* column checkbox.
- V. **Generate Web Service** - populate the Web Service Security window that manages the synching process with the Mobile app.
- VI. **Generate Sequence Mobile** - generates the document sequence numbering during operations from the mobile app.



There is also the heavily use **Web Services Security** window which is added upon by SFAndroid. Now we will look at each of these items in detail.

## Initial Load Window

We now look at how the **Initial Load** window is used for the first synchronising action from the mobile app.

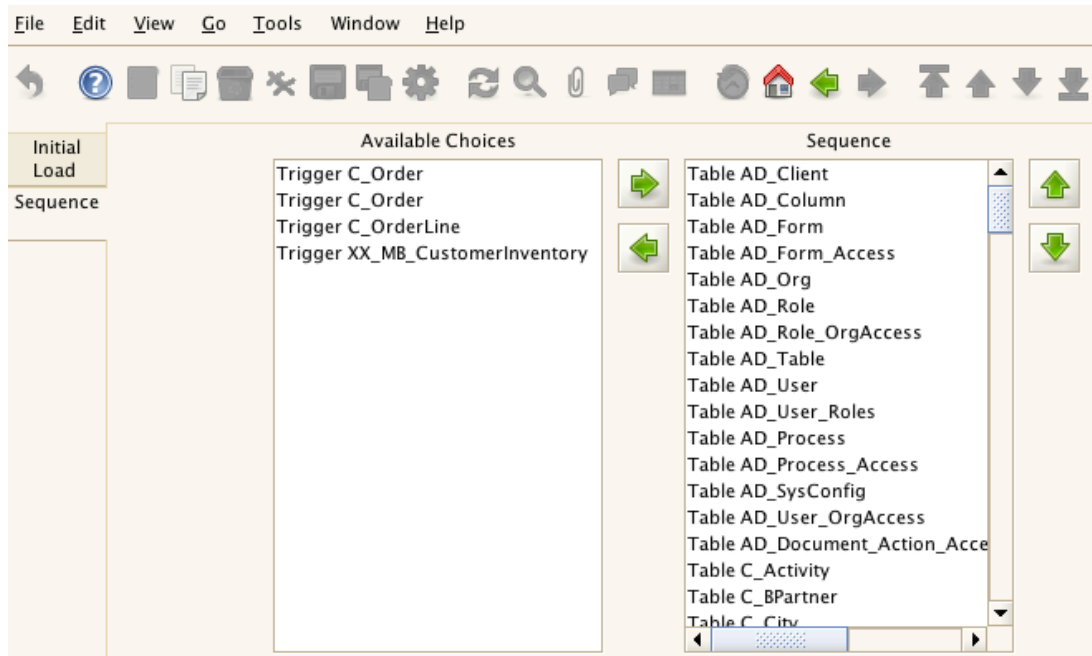
The screenshot shows the 'Initial Load' window for a specific table. The 'Client' is 'GardenWorld' and the 'Organization' is '\*'. The 'Name' is 'Table AD\_Client' and the 'Sequence' is '10'. The 'Description' is 'Create Table AD\_Client'. The 'SQL' field contains the script: `CREATE TABLE AD_Client ( AD_Client_ID NUMERIC(10) NOT NULL, Name VARCHAR(60) NOT NULL`. The 'Synchronizing Type' is 'Initial' and the 'Active' checkbox is checked. The 'Initial Load Movil System' is '1000024'.

When you open the Initial Load window, you will find that it is already populated with records. These are standard setup meta-data to populate the Mobile App. In the window you will find a **Synchronizing Type** that is 'Initial'.

| File Edit View Go Tools Window Help |                                 |          |                                        |                    |
|-------------------------------------|---------------------------------|----------|----------------------------------------|--------------------|
| Initial Load                        | Name                            | Sequence | Description                            | Synchronizing Type |
| Sequence                            | Table AD_User                   | 90       | Create Table AD_User                   | Initial            |
|                                     | Table AD_User_Roles             | 100      | Create Table AD_User_Roles             | Initial            |
|                                     | Table AD_Process                | 110      | CREATE TABLE AD_Process                | Initial            |
|                                     | Table AD_Process_Access         | 120      | CREATE TABLE AD_Process_Access         | Initial            |
|                                     | Table AD_SysConfig              | 130      | Create Table AD_SysConfig              | Initial            |
|                                     | Table AD_User_OrgAccess         | 140      | CREATE TABLE AD_User_OrgAccess         | Initial            |
|                                     | Table AD_Document_Action_Access | 150      | CREATE TABLE AD_Document_Action_Access | Initial            |
|                                     | Table C_Activity                | 160      | Create Table C_Activity                | Initial            |
|                                     | Table C_BPartner                | 170      | Create Table C_BPartner                | Initial            |
|                                     | Table C_City                    | 180      | Create Table C_City                    | Initial            |
|                                     | Table C_Region                  | 190      | Create Table C_Region                  | Initial            |
|                                     | Table C_Location                | 200      | Create Table C_Location                | Initial            |
|                                     | Table C_BPartner_Location       | 210      | Create Table C_BPartner_Location       | Initial            |
|                                     | Table C_Bank                    | 220      | Create Table C_Bank                    | Initial            |
|                                     | Table C_BankAccount             | 230      | Create Table C_BankAccount             | Initial            |

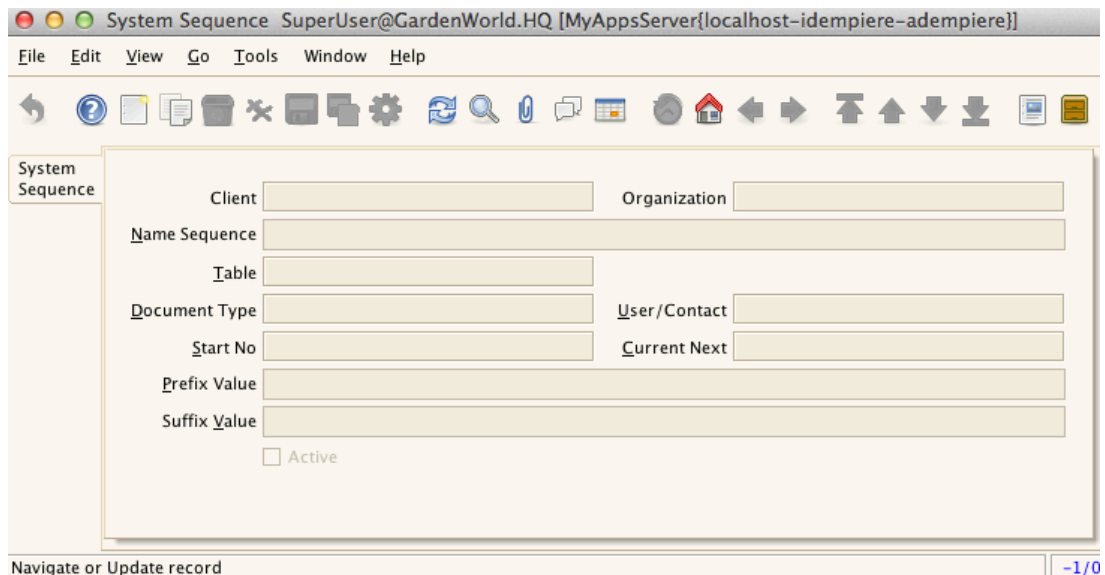
They list the tables that are to be created from scratch at the Mobile App side during Initial Load synchronizing. There is a SQL field that stores the exact SQL script to do that. The Mobile App will pick up from here and run those scripts into an *SQL-Lite* database and store them inside the mobile phone. This happens during synchronising from the phone.

## Initial Load - Sequence



The next tab in the *Initial Load* window is for **Sequence**, setting each script to execute in the order displayed. Again, this is already done for users in the window.

## System Sequence



Here you have the option to set the **Document Type** sequence numbering control. There are none set at the moment.

## Menu Mobile System

| Menu         | Name                    | Description                                       | Action | Special Form           | Table                            |
|--------------|-------------------------|---------------------------------------------------|--------|------------------------|----------------------------------|
| Mobil System | Outstanding Balances    | Displays Outstanding Balances by Customer...      | Report |                        | XX_RV_MB_OpenItem_XX_RV_MB_      |
|              | Visit Planning          | Sample Day Planning                               | Form   | MB Menu Planning Visit | XX_MB_PlanningVisit_Planning Vis |
|              | Sales Volume by Brand   | Shows the number of Units Sold by Make A...       | Report |                        | XX_RV_MB_MarksSold_Marks Sold    |
|              | Sales Volume by Product | Shows the number of Units Sold by Product         | Report |                        | XX_RV_MB_ProductsSold_Product    |
|              | Clients                 | Displays a list of clients assigned to Route S... | Form   | MB Clientes            | C_BPartner_Business Partner      |
|              | Current collection      | Shows and Collections Management Upgrad...        | Form   | MB Collection          | C_Cash_Cash Journal              |
|              | Made Orders             | Displays Made in the Day Orders                   | Form   | MB Order               | C_Order_Order                    |
|              | Outstanding Balances    | Shows Customers Outstanding Balances              | Form   |                        | C_Invoice_Invoice                |
|              | Inventories Made        | Inventories Displays Made in the Day              | Form   | MB Inventory           | XX_MB_CustomerInventory_Cust     |
|              | Order By Complete       | Displays Complete Pending Requests for the...     | Form   | MB Order               | C_Order_Order                    |
|              | Bank Deposit            | Create or update Bank Deposits                    | Form   | MB Deposit             | C_Payment_Payment                |
|              | RMA Day                 | Day Return Authorization                          | Form   | MB RMA                 | M_RMA_RMA                        |
|              | Visits of the Day       | Made in the Day Visits' View                      | Form   | MB Visit               | XX_MB_Visit_Visit to Customer    |
|              | Collection of the Day   | Displays a list of the Collections Made           | Form   | MB Collection          | C_Cash_Cash Journal              |
|              | Day Bank Deposits       | Displays a list of Bank Deposits Made             | Form   | MB Deposit             | C_Payment_Payment                |
|              | Questions               | Questions of the day, Sales, Effectiveness etc.   | Report |                        |                                  |
|              | Synchronization         | Establishes communication with ADeMPIere...       | Form   | MB Synchronization     |                                  |
|              | About                   | Software Manufacturer Information                 | Form   | MB About               |                                  |

This window defines the arrangement of the menu, its Action types and settings to operate on the Android App or the mobile phone side. There are 18 items setup. The three Action types are Form, Process, and Report.

|                         |                        |                                            |               |                        |
|-------------------------|------------------------|--------------------------------------------|---------------|------------------------|
| Menu<br>Mobil System    | Client                 | GardenWorld                                | Organization  | *                      |
|                         | Name                   | Visit Planning                             |               |                        |
|                         | Description            | Sample Day Planning                        |               |                        |
|                         | Action                 | Form                                       | Special Form  | MB Menu Planning Visit |
|                         | Table                  | XX_MB_PlanningVisit_Planning Visi          |               |                        |
|                         | Image                  | klipper_h                                  |               |                        |
|                         | Sql WHERE              |                                            |               |                        |
|                         | Group By Clause        |                                            |               |                        |
|                         | Sql ORDER BY           | XX_MB_PlanningVisit.SeqNo                  |               |                        |
|                         | Activity Type          | Menu                                       | Sequence      |                        |
|                         |                        | <input type="checkbox"/> Summary Level     |               |                        |
|                         | Menu Type              | Menu                                       | Read Write    |                        |
|                         |                        | <input checked="" type="checkbox"/> Active | Insert Record |                        |
|                         | Rule After Web Service |                                            |               |                        |
|                         | Name of Image Error    |                                            |               |                        |
| Web Service Type        |                        |                                            |               |                        |
| Rule Before Web Service |                        |                                            |               |                        |

Looking at an example above, *Visit Planning*, you can see that its **Action** is for a **Form**, in which the **Special Form** is set to *MB Menu Planning Visit*. That is using the **AD** standard where in the Form window, if you open as **SystemAdmin** to take a look below describes the

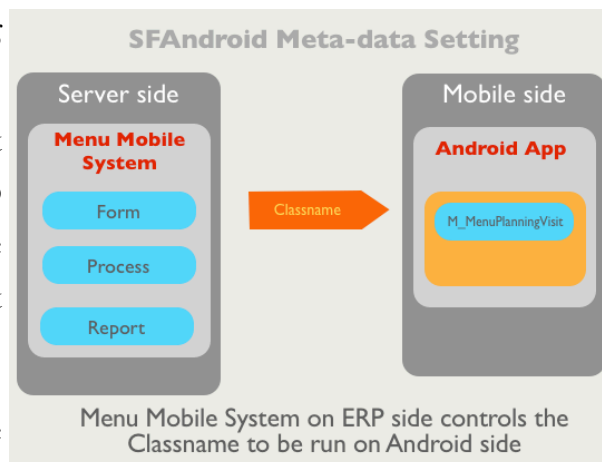
Java class it is calling. This is going to happen not on the ERP side but the mobile app side. The use of AD meta-data approach allows for more reusability of the AD and better future maintenance and extensibility as well as a best standard practice.

The screenshot shows a web-based form for configuring SFAndroid meta-data. The form is titled 'Form' and 'Access'. It contains the following fields:

- Client:** System
- Organization:** (empty)
- Name:** MB Menu Planning Visit
- Description:** Menus Displayed by clicking on the icon Planning to Visit
- Comment/Help:** (empty)
- Active:** ☒
- Entity Type:** Android
- Data Access Level:** Client+Organization
- Classname:** org.appd.view.M\_MenuPlanningVisit
- Context Help:** (empty)

Here in the *Classname* field it is calling *org.appd.view.M\_MenuPlanningVisit*. This class is an Android Java class and it exists at the mobile side. When the app is downloaded into your mobile phone it must contain this class in its jar or it will hit a *ClassNotFoundException*.

This also allow better security where unauthorised code cannot run on your mobile app without been defined here on the ERP server side.



## Mobile Other Action

Besides the Form action, the Mobile Menu can also contain other actions such as Process and Report. The screenshot below is for a Process example.

The screenshot shows a web-based configuration interface for a Mobile Menu. The interface has a menu bar (File, Edit, View, Go, Tools, Window, Help) and a toolbar with various icons. On the left, there is a sidebar with 'Menu' and 'Mobil System'. The main area contains a form for configuring a menu item. The form fields are as follows:

- Client:** GardenWorld
- Organization:** \*
- Name:** Outstanding Balances
- Description:** Displays Outstanding Balances by Customer and Delivery Address
- Action:** Report
- Table:** XX\_RV\_MB\_OpenItem\_XX\_RV\_MB\_OpenItem
- Special Form:**
- Process:** Open Item\_inf\_mb\_OpenItem
- Image:** payment\_mixed\_h
- Sql WHERE:**
- Group By Clause:**
- Sql ORDER BY:** XX\_RV\_MB\_OpenItem.Customer
- Sequence:** 1
- Summary Level:** ☐
- Menu Type:** Menu
- Read Write:**
- Active:** ☒
- Insert Record:**

Likewise this follows the *Compiere* convention of defining a *Report and Process* item which in this case is '*Open Item\_inf\_mb\_OpenItem*'. You also specify the table it will be accessing in the SQL-Lite database on the mobile phone. You can also determine the icon image it be displaying for this item on the menu.

Again, all this is happening at the mobile side of the code which will handle its actual rendition or output the Android way.

## Menu Mobile Synchronization

This menu has the same records as Menu Mobile System just that it is flagged by the column *Menu* = 'N' which indicates it is for table data synchronization between the mobile and the server.

## Generate Web Service

Web services window is used to store settings which manage the mobile app syncing process. This process handles any new setting needed. Most of the web services required by the mobile app are already generated.

The nature of the mobile app is a standalone ERP system that handles its own process and merely replicate back key information back to the server. Thus what is more needed is just Read or CreateUpdate WS routines.

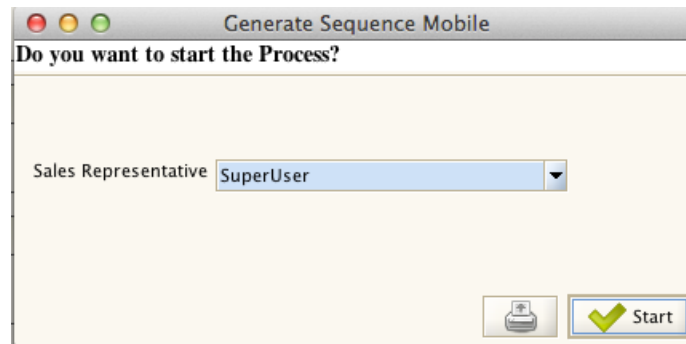
## Web Service Security Window

This window is accessed via the GardenAdmin client where you can examine the defined settings for mobile-server synchronization management. There are some new fields introduced (not shown in screen) that helps the synch activity.

| Web Service Type         | Web Service Type | Search Key              | Name                       | Table                                        |
|--------------------------|------------------|-------------------------|----------------------------|----------------------------------------------|
| Web Service Parameters   | 1                | ReadDays                | Read Business Days         | XX_RV_MB_Day_View Of Mobile Days             |
|                          | 27               | CreateVisitLine         | Create Online Business     | XX_MB_VisitLine_Line of Visit                |
| Web Service Field Input  | 28               | CreateCustomerInventory | Customer create inventory  | XX_MB_CustomerInventory_Customer Inven.      |
|                          | 2                | CreateDay               | Insert In Days             | XX_MB_Day_Visiting Day                       |
| Web Service Field Output | 31               | UpdateSequence          | Update Sequence            | XX_MB_Sequence_XX_MB_Sequence (Sequen.       |
|                          | 34               | ReadC_City              | Read City                  | XX_RV_MB_C_City_XX_RV_MB_C_City              |
| Web Service Access       | 33               | ReadC_Region            | Read Region                | XX_RV_MB_C_Region_XX_RV_MB_C_Region          |
|                          | 36               | ReadRMAType             | Read RMA Type              | XX_RV_MB_M_RMAType_XX_RV_MB_M_RMA..          |
|                          | 37               | ReadInOut               | Read Delivery              | XX_RV_MB_M_InOut_XX_RV_MB_M_InOut            |
|                          | 38               | ReadInOutLine           | Read Delivery Line         | XX_RV_MB_M_InOutLine_XX_RV_MB_M_InOu.        |
|                          | 41               | CreateRMA               | Read RMA                   | M_RMA_RMA                                    |
|                          | 42               | CreateRMALine           | Create RMA Line            | M_RMALine_RMA Line                           |
|                          | 19               | ReadSequences           | Read Sequence              | XX_RV_MB_Sequence_View of Mobile Sequen.     |
|                          | 30               | ReadInitialLoad         | Read Data Update           | XX_MB_InitialLoad_Initial Load Mobile System |
|                          | 46               | ReadTables              | Read Tables                | XX_RV_MB_AD_Table_View Of Tables for Mo..    |
|                          | 47               | ReadColumns             | Read Columns               | XX_RV_MB_Column_XX_RV_MB_Column              |
|                          | 24               | CreateOrder             | Create Sales Order         | C_Order_Order                                |
|                          | 44               | ReadADForm              | Read Form                  | XX_RV_MB_AD_Form_View Of Mobile Form         |
|                          | 55               | ReadMenuList            | Read Menu System and Sy... | XX_RV_MB_MenuList_View of Menu List          |

## Generate Sequence Mobile

This sequence generating happens during operations where documents are created such as those in use by the Sales Representative. Examples are Sales Orders and Customers' Inventories.



**June 2** Canoa Quebrada, BRAZIL. After a wild dune ride during the day, we settled down for an Italian dinner among friends. Left-most is Ricardo Santana of Sao Paulo, from KENOS, the main contributor company for Localization Brazil. Next to him is Edilson Neto of Fortaleza.



## SALES FORCE MODULE

Before we look at how to set the above all up, we look at the Sales Force module itself but from technical systems point of view. The concept here is that Salesforce is a vertical application and the SFAndroid is meant to be a horizontal framework to build other verticals on top of it.

### Concept of Verticals

As explained earlier, ERP is a horizontal application and Compiere right through iDempiere play this concept very well with the Application Dictionary framework which allows changes to be made without change to code and a high reusability of its ready models and APIs. SFAndroid similarly is an extension of that idea into the mobile world of enterprise business.

SFAndroid application comes already coupled with a vertical which is *SalesForce* of which the ‘SF’ characters in the project name stands for. Salesforce if some of you may know is the name of a popular SAP SaaS (Software as a Service) project that tries to automate sales-force activity and provide that service from a *Cloud* system to avoid the users to do any setup. Needless to say Salesforce.com face a huge challenge as business is very complex and the infrastructure is not exactly there yet.

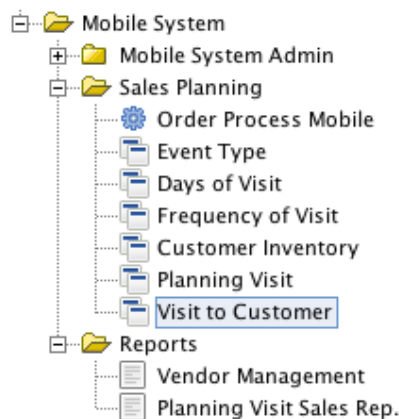
Thus, such a stand-alone application to do Salesforce is welcome in this form. Been a difficult vertical, the idea of vertical reusing a good horizontal framework is what this project is showing in a big way. Readers can appreciate how this is achieved here and reflect about their own vertical.

### SalesForce Menu

The Sales Force module designed by the contributor company, **ERP CONSULTORES Y ASOCIADOS** and **- LUBO SISTEMAS** of **VENEZUELA** is for the particular use of their client, **AGROSILOS** of **PANAMA**. Thus this is like the GardenWorld example in Compiere, where it acts as a ready-made sample for users with similar requirements. However it will not be an exact fit for many companies and thus a training stint or an outright outsource contract to the contributors above is recommended. They be the best as they know their own code. Also, for their sharing spirit, they deserve to be referred to first be-

fore your own local IT provider. They are also open to work with your local vendor, and this is where minimally a training stint is a good option. In case you need my assistance on that or to assist in an English version training, I can help arrange for your needs. As it is, this ‘GardenWorld’ sample is a great way to learn how to extend the app.

Originally much of the menu items in Mobile System Admin were in Spanish and I converted them to English using *GoogleTranslator* and a bit of help from the team in Venezuela. Nevertheless some of the items may not be accurately conveyed. Please drop me a note if you got a better translation from some of the obscure terms or functions. The graphical display and code on the mobile side can be of assistance and eventually via open source we can solve all mysteries of life.



If you open up some of the items in the Sales Planning menu you will see basic data is already present. The ERP side is for central management to derive reports and thus the Reports menu of Vendor Management and Planning Visit Sales Rep. is aimed at.

We will not be explaining this vertical use in exhaustive detail as this document is more of a technical guide. Contribution of further document in the form of a user manual to show how this vertical fully operates is appreciated. Later section will just give enough treatment how to get your mobile app running and pointing to your server-side.

## SETTING UP

### Direct Hot Setup

The SFAndroid-server is now converted into a full OSGi plugin for installing into the iDempiere stack. Note that the jar timestamp may differ as found in

<https://sourceforge.net/projects/redt/files/p2/SFAndroid/> . You can install it directly from inside your running iDempiere ‘ss’ console.

install [http://downloads.sourceforge.net/project/redt/p2/SFAndroid/org.idempiere.SFAndroid-Server\\_1.0.0.<timestamp>.jar](http://downloads.sourceforge.net/project/redt/p2/SFAndroid/org.idempiere.SFAndroid-Server_1.0.0.<timestamp>.jar)

You can then apply this 2Pack from the same repository.

- MobileData.zip

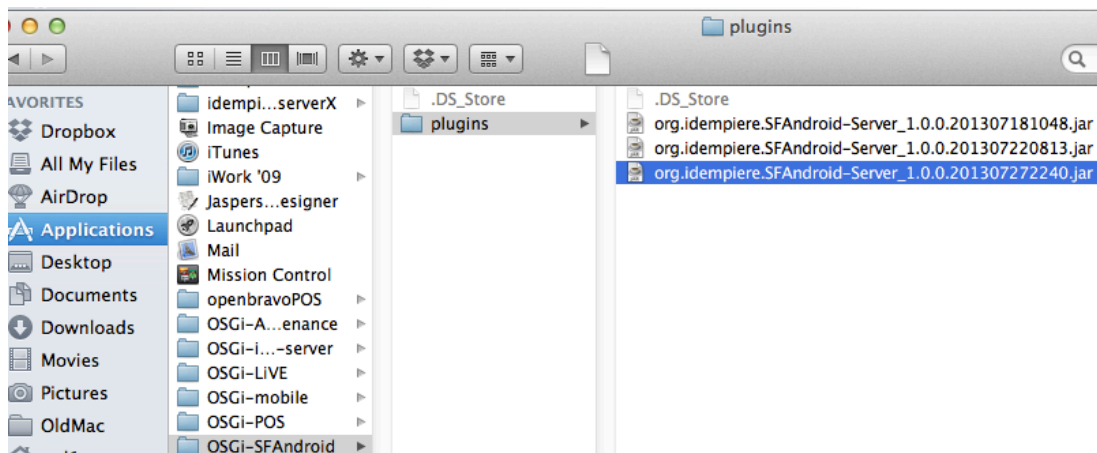
This 2Pack is to be applied when you login as GardenAdmin client.

The server plugin jar’s 2Pack (MobileSynch.zip) will automatically deploy to define the above meta-data and content for the operations described here.

Though the above is sufficient for an experienced implementor, the following pages guide you through some setup process in more elaborate detail.

### Downloading the SFAndroidServer Plugin

You can download from the link above, the latest SFAndroid-Server<version>.jar and install it from your server. This is also useful for the *Felix Console* method of installation.



The screen-shot here shows the plugin downloaded into Applications/OSGi-SFAndroid/plugins.

Note that from time to time the plugin version will be updated by me when there are changes to it in the bitbucket source. You then have to uninstall your present plugin and install again the new plugin.

## At the OSGi Console

Now you startup your iDempiere-Server. Once it has fully launched, you type in 'ss' into the terminal box to see the stack of plugins and you will note that the last one does not show your plugin. Enter the install statement will change that (timestamp of jar may differ).

```

363  RESOLVED    org.adempiere.webstore.servlet_1.0.0.v20130712-1409
364  ACTIVE      org.apache.ecs_1.0.0.v20130712-1409
365  <<LAZY>>    org.compiere.db.oracle.provider_1.0.0.v20130712-1409
366  ACTIVE      org.compiere.db.postgresql.provider_1.0.0.v20130712-1409
367  ACTIVE      org.idempiere.felix.webconsole_1.0.0.v20130712-1409
368  ACTIVE      org.idempiere.hazelcast.service_1.0.0.v20130712-1409
          Fragments=146
369  ACTIVE      org.idempiere.webservices_1.0.0.v20130712-1409
370  ACTIVE      org.zkoss.zk.library_6.0.0.v20130712-1409
373  RESOLVED    javax.el_2.2.0.v201105051105

osgi> install file:/Applications/OSGi-SFAndroid/plugins/org.idempiere.SFAndroid-Server_1.0.0.201307272240.jar
Bundle id is 376

osgi>

```

The console returned in this example: Bundle id is 376. You then start 376 to let the embedded 2Pack deploy. It will take some minutes depending on the speed of your server.

```

idempiere-server — java — 133x43
01:35:14.829 PackInHandler.processDeferElements: Processing Element: AD_Menu - table [43]
01:35:14.829 PackInHandler.processDeferElements: Imported Defer Element: AD_Menu - table [43]
01:35:14.829 PackInHandler.processDeferElements: Processing Element: AD_Menu - table [43]
01:35:14.848 PackInHandler.processDeferElements: Processing Element: AD_Menu - table [43]
01:35:14.848 PackInHandler.processDeferElements: Imported Defer Element: AD_Menu - table [43]
01:35:14.848 PackInHandler.processDeferElements: Processing Element: AD_Menu - table [43]
01:35:14.893 PackInHandler.processDeferElements: Processing Element: AD_Menu - table [43]
01:35:14.893 PackInHandler.processDeferElements: Imported Defer Element: AD_Menu - table [43]
01:35:14.900 Trx.commit: **** P0SaveAD_Package_Imp_ef432b36-c94b-49c6-ae41-9e9fd14bf5bc [43]
01:35:14.907 Trx.commit: **** P0SaveAD_Package_Imp_In_6a0472cb-09c5-44b9-b815-60216ccc59af [43]
01:35:20.198 PackIn.importXML: End Parser [43]
01:35:20.252 Trx.commit: **** Trx_4703ec34-c789-417c-93b4-e7c4d26d6e05 [43]
01:35:20.252 PipDictionaryService.merge: commit Trx_4703ec34-c789-417c-93b4-e7c4d26d6e05 [43]
org.idempiere.SFAndroid-Server 1.0.0 installed.
01:39:23.695 Login.getClients: User=SuperUser [28]
01:39:23.801 Trx.commit: **** P0SaveAD_User_cc0cf480-dc37-462b-879d-66af8fe0e7f5 [28]
01:39:23.991 MRole.get: AD_Role_ID=0, AD_User_ID=0, reload=false [28]
01:39:24.002 MRole.get: AD_Role_ID=0, AD_User_ID=0, reload=false [28]

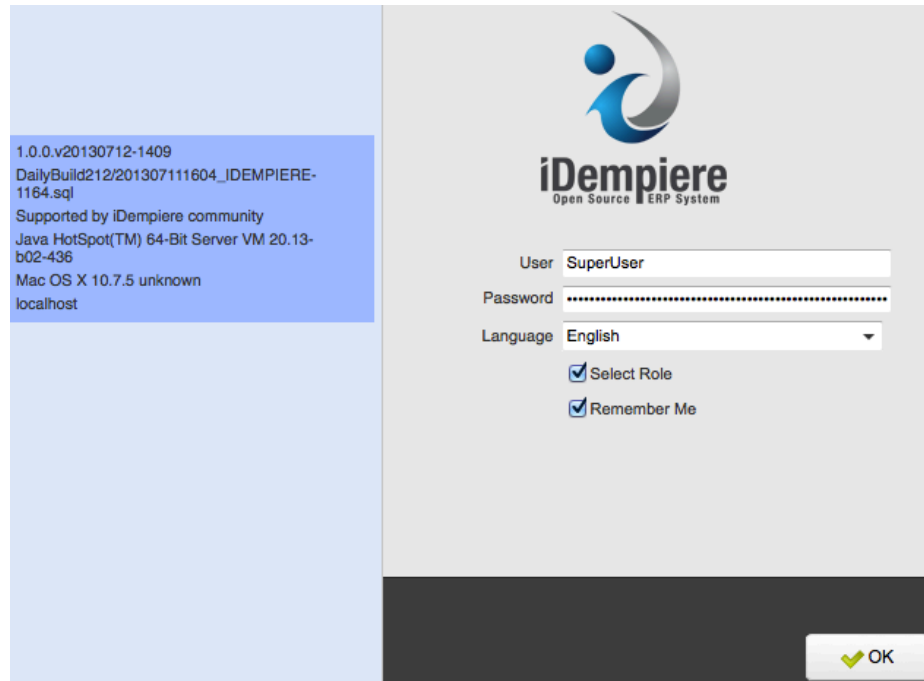
```

At the end of Pack-In, you can see org.idempiere.SFAndroid-Server 1.0.0 installed.

The Pack-In log lines may only appear if your *Preferences* is set to *Fine* in your iDempiere System. So please understand if you see nothing and the server is processing a lot.

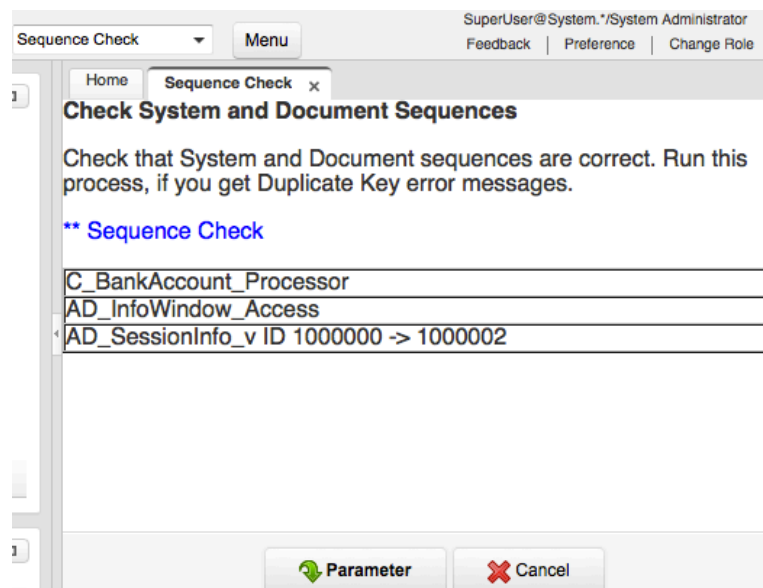
## Sequence Check

At your browser, you then go to your login screen with <http://localhost:8080/webui>



The image shows the iDempiere login interface. On the left, a blue box contains system information: 1.0.0.v20130712-1409, DailyBuild212/201307111604\_IDEMPHERE-1164.sql, Supported by iDempiere community, Java HotSpot(TM) 64-Bit Server VM 20.13-b02-436, Mac OS X 10.7.5 unknown, localhost. The main area features the iDempiere logo and the text 'Open Source ERP System'. Below this are input fields for 'User' (containing 'SuperUser'), 'Password' (masked with dots), and 'Language' (set to 'English'). There are two checked checkboxes: 'Select Role' and 'Remember Me'. An 'OK' button with a green checkmark is at the bottom right.

You tick Select Role so that you can select System to carry out the Sequence Check of tables' primary sequence numbering to avoid conflict in ID during record processing. Usually you need not Select Role if your client is default to System. At the System Menu, you carry out a *Sequence Check*. It should return you some lines of processed sequences.



The image shows the 'Sequence Check' window in the iDempiere application. The title bar indicates the user is 'SuperUser@System.\*System Administrator'. The window has a 'Menu' button and a 'Sequence Check' dropdown. The main content area is titled 'Check System and Document Sequences' and contains the text: 'Check that System and Document sequences are correct. Run this process, if you get Duplicate Key error messages.' Below this is a section labeled '\*\* Sequence Check' with a table of results:

|                                        |
|----------------------------------------|
| C_BankAccount_Processor                |
| AD_InfoWindow_Access                   |
| AD_SessionInfo_v ID 1000000 -> 1000002 |

At the bottom, there are two buttons: 'Parameter' (with a green arrow icon) and 'Cancel' (with a red X icon).

## Mobile Data Pack-In

Now you are ready to pack in the mobile server-side setup data into the new models that were packed in during plugin install. Change the **Role** you are in to *GardenAdmin*. Go to the **Pack In** menu.

The screenshot shows the iDempiere web application interface. At the top, the user is logged in as 'SuperUser@GardenWorld.\*GardenWorld Admin'. The 'Pack In' menu is selected. The main form contains the following fields:

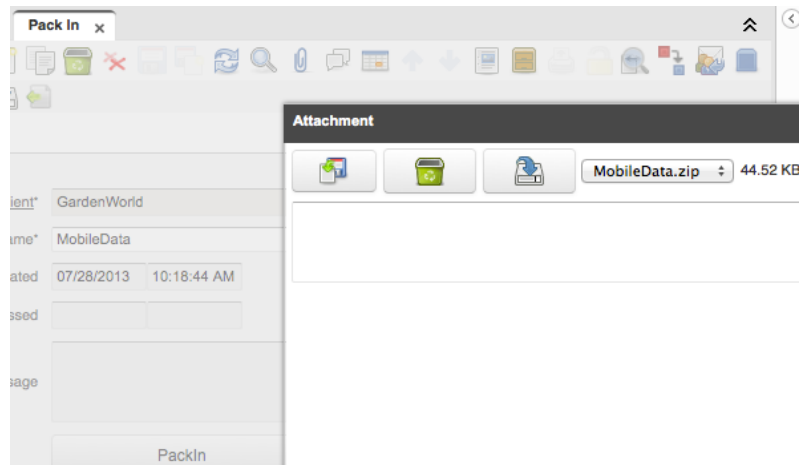
- Client\***: System
- Organization\***: \*
- Name\***: org.idempiere.SFAndroid-Server
- Created**: 07/28/2013 9:54:52 AM
- ☐ Update System Maintained Application Dictionary
- Date Processed**: 07/28/2013 9:56:00 AM
- Process Message**: Processed=2015 Un-Resolved=0
- PackIn** button

You will notice that the earlier system pack-in record is logged with objects processed\* and non Un-Resolved. This is double confirmation that everything is good thus far. Now, if for any reason, you do not wish to pack-in the system model via plugin you can do so manually from MobileSynch.zip also found in my SourceForge.net repository.

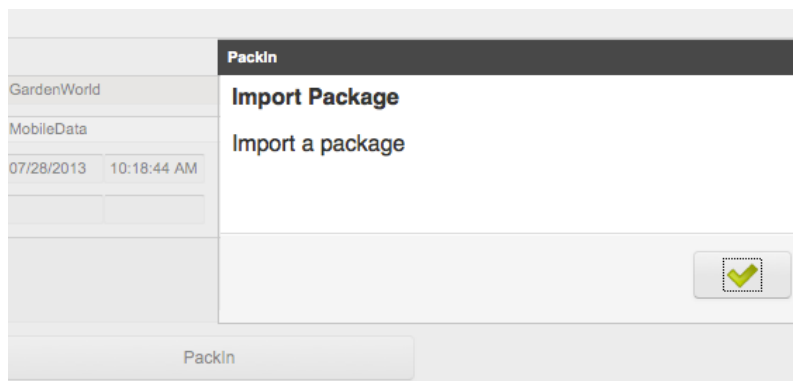
Click on the **New** record icon so that you can select a new 2Pack zip file for pack in. Give it a **Name**, such as 'MobileData' and save it. Then click on **Attachment** (paper clip icon) to select the MobileData.zip downloaded from my repository here:

<http://sourceforge.net/projects/red1/files/p2/SFAndroid/MobileData.zip/download>

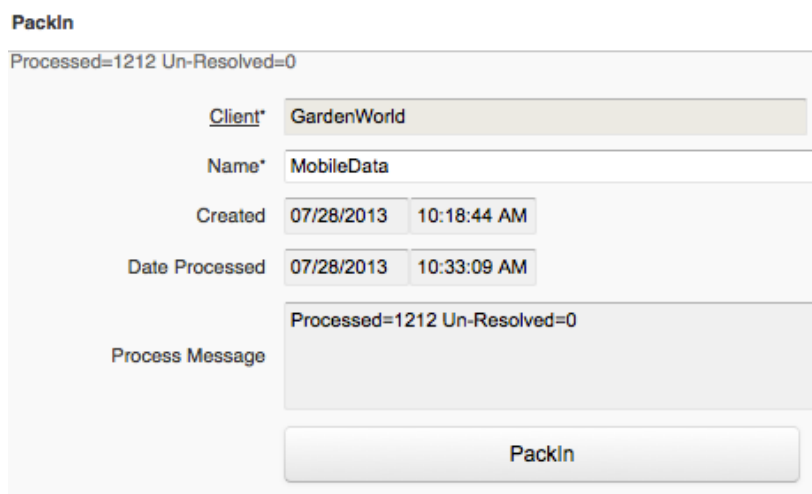
\*Note that the number of objects processed may differ when I upload the latest artifacts.



Search inside your server for zip file and begin the **Import Package** process.



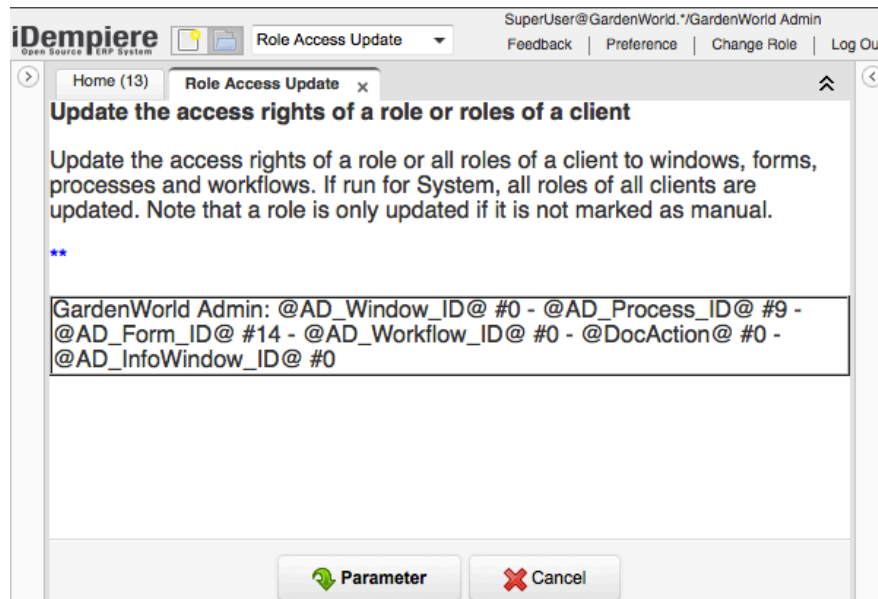
You should finally see the results as shown here. With 1212 records processed and zero Un-Resolved errors. If there are any errors take note of the logs and the last line should show



you which record and its details and which field is unresolved.

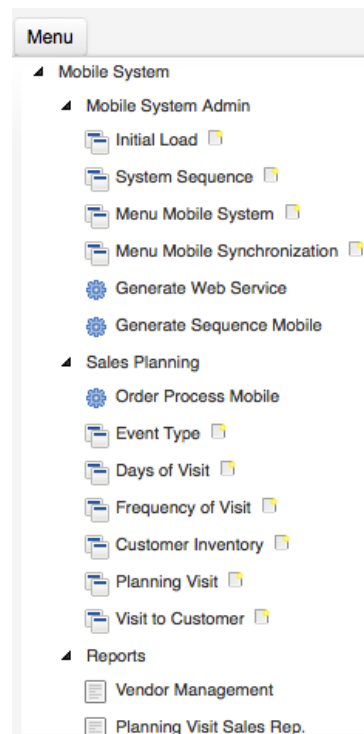
## Role Access Update

What is left now is the **Role Access Update**. This is a common practice explained in many documents before, so no further explanation given here. The screen below shows a successful update.



Total of 9 Processes, and 14 Forms are updated for the GardenAdmin role to access.

Next, you do a **Logout** and **Login** and you can see the full model of the SFAndroid server-side setup and configuration windows as shown on the right. You can click on the Menu Mobile System or Menu Mobile Synchronization to see the contents.



## DEVELOPER'S NOTE

### BitBucket Source

The source code with latest 2Packs for server-side are committed to

<https://bitbucket.org/red1/org.idempiere.sfandroid-server>

Further notes later will cater for the WS and mobile side and assist the developer to work on the code and contribute back to minimize mutual cost of maintenance and sustenance.

### SourceForge.net Repository

This repository contains all the files referred to including this manual guide. It is for easy reference and downloading for use, testing and implementation. The URL to get this is <https://sourceforge.net/projects/red1/files/p2/SFAndroid/> (screen below is from v1.6)

|                                                                       |
|-----------------------------------------------------------------------|
| <a href="#">↑ Parent folder</a>                                       |
| <a href="#">ExpDat20130812_190754.jar</a>                             |
| <a href="#">MobileData.zip</a>                                        |
| <a href="#">MobileSynch.zip</a>                                       |
| <a href="#">org.idempiere.SFAndroid-Server_1.0.0.201308122100.jar</a> |
| <a href="#">org.idempiere.SFAndroidWS_1.0.0.201308102245.jar</a>      |
| <a href="#">SFAndroid.pdf</a>                                         |
| <a href="#">ExpDat_ES.jar</a>                                         |
| <a href="#">MobileData_ES.zip</a>                                     |
| <b>Totals: 8 Items</b>                                                |

The server-side plugin is [org.idempiere.SFAndroid-Server<time-stamp>.jar](#) and the Web-Services plugin is [org.idempiere.SFAndroidWS<time-stamp>.jar](#).

The [MobileSynch.zip](#) is the system model 2Pack that is already embedded into the plugin jar. There is the original Latin-American (Spanish) version of the sample data and it is [MobileData\\_ES.zip](#). The [MobileData.zip](#) is translated by me and I appreciate review on that. The database dump that contains the MobileSynch and MobileData\_ES Pack-Out information is [ExpDat\\_ES.jar](#).

Thus the ExpDat<timestamp>.jar is a fully English translated version. I am using that to maintain or modify the 2Packs. You can also do the same after DBRestore from this DB dump inside the ExpDat jar. Remember to DBExport back a new jar and you can send it back to me if there is any extensions or improvements you did. I would appreciate some README.txt to explain what you did for my review.

I will not maintain the original ExpDat\_ES.jar and it may break if the system model in MobileSynch changes. I will interact with the Venezuelan team if they are following the original Compiere approach of handling translation through Language packs. They told me somewhat they do but I have yet to go into that.

## 2Pack Exporting

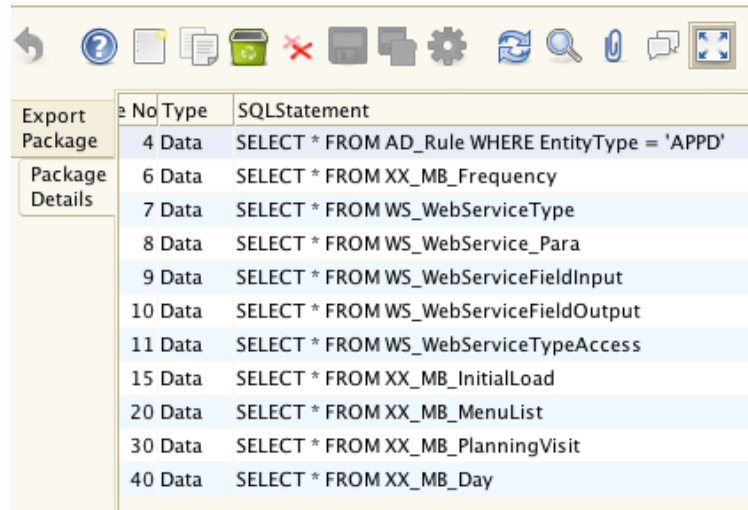
From the ExpDat<time-stamp>.jar you can restore to your present iDempiere and examine the Pack Out contents. Below is for the System side (screen below is from v1.6):

| Export Package  | ie No | Type           | Entity Type            | Menu                 | SQLStatement                                        |
|-----------------|-------|----------------|------------------------|----------------------|-----------------------------------------------------|
| Package Details | 20    | Entity Type    | Android                |                      |                                                     |
|                 | 30    | Entity Type    | Lubo Sistemas (LS)     |                      |                                                     |
|                 | 40    | Entity Type    | Sistemas Globales (SG) |                      |                                                     |
|                 | 42    | Data           |                        |                      | SELECT * FROM AD_Element WHERE EntityType = 'APPD'  |
|                 | 45    | Data           |                        |                      | SELECT * FROM AD_Column WHERE EntityType = 'APPD'   |
|                 | 50    | Data           |                        |                      | SELECT * FROM AD_Table WHERE EntityType = 'APPD'    |
|                 | 53    | Data           |                        |                      | SELECT * FROM AD_TableIndex WHERE Name Like 'xx%'   |
|                 | 60    | Data           |                        |                      | SELECT * FROM AD_Val_Rule WHERE EntityType = 'APPD' |
|                 | 62    | Data           |                        |                      | SELECT * FROM AD_Form WHERE EntityType = 'APPD'     |
|                 | 64    | Data           |                        |                      | SELECT * FROM AD_Process WHERE EntityType = 'APPD'  |
|                 | 66    | Data           |                        |                      | SELECT * FROM AD_Process WHERE EntityType = 'LS'    |
|                 | 68    | Table          |                        |                      |                                                     |
|                 | 70    | Application... |                        | Web Service Security |                                                     |
|                 | 80    | Application... |                        | Mobile System        |                                                     |

The dependencies are exported first. They are the new EntityTypes APPD, LS, and SG. Then certain records for Element, Column, Tables, Validation Rules, Forms, and Processes created for the new models. Finally the Menu tree of Web Service Security and Mobile System are exported out. This whole package is tested to pack back well in the present iDempiere i.o.c. Note that if you restore this database at later dates, you may need to apply the latest migration scripts in between the time-stamp and present date. (You may even have to go back few more days from the time-stamp possibly when I do not update that sudden when I am working on this).

## GardenWorld Sample

The GardenWorld client *Pack In* only works when you are in the GardenWorld menu due to role access security control. Similarly it is also packed out from inside the client. So login as GardenAdmin to view the *Pack Out* contents below.



| Package No | Type | SQLStatement                                    |
|------------|------|-------------------------------------------------|
| 4          | Data | SELECT * FROM AD_Rule WHERE EntityType = 'APPD' |
| 6          | Data | SELECT * FROM XX_MB_Frequency                   |
| 7          | Data | SELECT * FROM WS_WebServiceType                 |
| 8          | Data | SELECT * FROM WS_WebService_Para                |
| 9          | Data | SELECT * FROM WS_WebServiceFieldInput           |
| 10         | Data | SELECT * FROM WS_WebServiceFieldOutput          |
| 11         | Data | SELECT * FROM WS_WebServiceTypeAccess           |
| 15         | Data | SELECT * FROM XX_MB_InitialLoad                 |
| 20         | Data | SELECT * FROM XX_MB_MenuList                    |
| 30         | Data | SELECT * FROM XX_MB_PlanningVisit               |
| 40         | Data | SELECT * FROM XX_MB_Day                         |

The many rules that are used to create table content is in the *AD\_Rule* table. But it was created manually by the developer and not within the AD. I have to change the role access of *AD\_Rule* to *System & Client* to allow **PackOut** access rights to its records. The *WS\_Web-service* data is for the new settings for the web-service in use by SFAndroid. The *XX\_MB\_Menulist* table content is for menu item *Menu Mobile System* (when menu field is checked) and *Menu Mobile Synchronization* (menu field unchecked).



**July 14, 2013.** Attending the wedding of Shunjiro Yatsuzuka and Mari Sato. Shunjiro is the head of ADempiere community in Japan. 3 persons by my right are also contributors to our project there. 2nd right is Daisuke Kubotta, who helps translated ADempiere into Nippon-Go, the local Japanese language. 3rd right is Yuichi Terada, head of Open Source division of the large Nomura Research Institute that is presently the most serious Open Source Software corporate effort in Japan. We are proposing for a Japan iDempiere Conference next Sakura Spring Festival, 2014 in Japan inviting our top contributors Carlos Ruiz and Low Heng Sin.

**July 9/10.** I was invited to Hiroshima to speak at a local college, visit the Mazda Museum and its assembly production line, the Peace Memorial, the old palace and stayed in the countryside. All with the compliments of Kensuke Ishibashi, member of the Japanese ADempiere Users Group.



## SFAndroid Web-Services

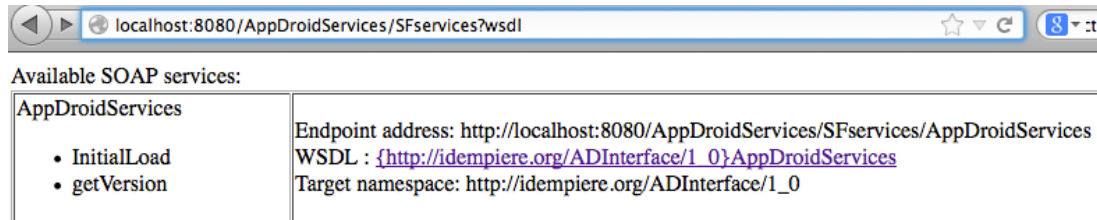
The web-services (WS) used by the SFAndroid team is done on the old ADempiere web-service module which in iDempiere has to be upgraded. With the help of Deepak Pansheriya, who created Composite Web-Services into the iDempiere core WS plugin, I managed to migrate the SFAndroids-3ews into `org.idempiere.SFAndroidWS.jar`. This plugin is made on latest iDempiere.1.0.c and dependent on `org.idempiere.webservices` plugin, which has to export out some packages (sent to iDempiere for merging).

It can be downloaded from my SF link above. When successfully started will give the following prompt at the console:

```
start 158
Aug 8, 2013 6:51:49 AM org.apache.catalina.core.ApplicationContext log
INFO: Initializing Spring root WebApplicationContext
Aug 8, 2013 6:51:53 AM org.compiere.util.Language getLanguage
INFO: Adding Language=pl, Country=PL, Locale=pl_PL
Aug 8, 2013 6:51:53 AM org.apache.cxf.service.factory.ReflectionServiceFactoryBean buildServiceFromClass
INFO: Creating Service {http://idempiere.org/ADInterface/1_0}AppDroidServices from class org.mb.appdroid.AppDroidServices
Aug 8, 2013 6:51:53 AM org.apache.cxf.endpoint.ServerImpl initDestination
INFO: Setting the server's publish address to be /AppDroidServices

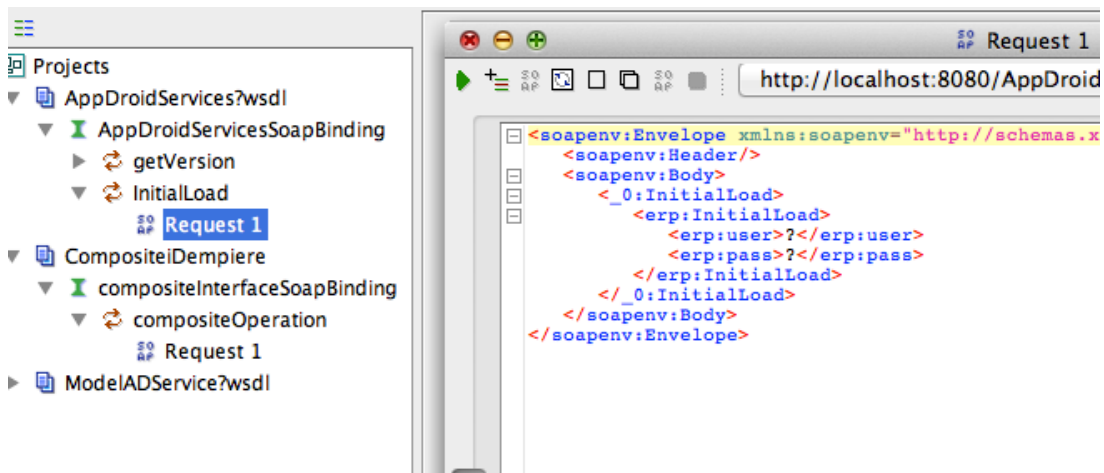
osgi>
```

You can then test it out by calling the URLs in the screen-shots below.



Available RESTful services:

<http://localhost:8080/AppDroidServices/SFservices/AppDroidServices?wsdl>



Later, the development team may refactor away this WS plugin and reuse the present and more efficient iDempiere webservice. However *SFAndroid WS* in essence is doing replication of synching between the server and the client and not realtime processing of documents, which the powerful *Compsosite WS* is all about.

## POWrapper

What is significant to relate to developers wishing to work on iDempiere source-code is about the way to extend or over-ride a core model class. The following is an example taught personally to me by Low Heng Sin.

The non-recommended way to extend is this, where you have a new generated X\_class and I\_class:

```
public class MClientInfo extends X_AD_ClientInfo
{
```

The recommended way to do so is this, where you do not depend on the X and I classes:

```
public class MClientInfo extends org.compiere.model.MClientInfo
{
    ...
}
```

The implementation of the extension call will thus be:

```
// Dixon Martinez Menu Mobile System
//      setAD_Tree_SF_SyncMenu_ID(SF_SyncMenu_ID);
//TODO example of hengsin - using POWrapper (extends core M<Class>)
I_AD_ClientInfo iinfo = POWrapper.create(this, I_AD_ClientInfo.class);
iinfo.setAD_Tree_SF_SyncMenu_ID(SF_SyncMenu_ID);
// End Dixon Martinez
//
```

Here you wrap only the extended I\_class without generating the X counterpart. You then packaged your new M class as a distinct package for your plugin own usage.

```
package org.compiere.mb.model;
```

You are thus not taking over the core MInfoClient and overriding it. Heng Sin explained that this minimizes or eliminates the future impact to the core breaking your module and vice versa.

## ANDROID DEVELOPMENT

The Android mobile app been a new introduction into the community and project deserves a starter tutorial for our developers on how to review and debug such code. Though the code can now run off the shelf as it is decoupled from the server-side which is now working in iDempiere OSGi framework, I dedicate some practical insight that will bring our developers to speed. Those attempting this should already be experienced project developers. I also do the minimal 'freedom' code to unlock the hard-coded backend URL to WS communication so that users can implement this freely on iDempiere or own server-side ERP.

### Using the ADK

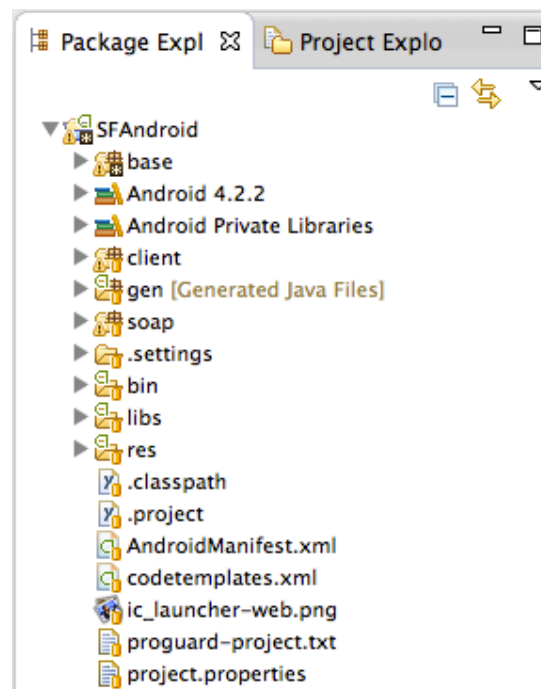
You should have the Android Development Kit (ADK) which is an Eclipse IDE for Android development setup on your own. Then you can check out the SFAndroid source from my bitbucket repository or from the Venezuelan team which I cloned and played around from.

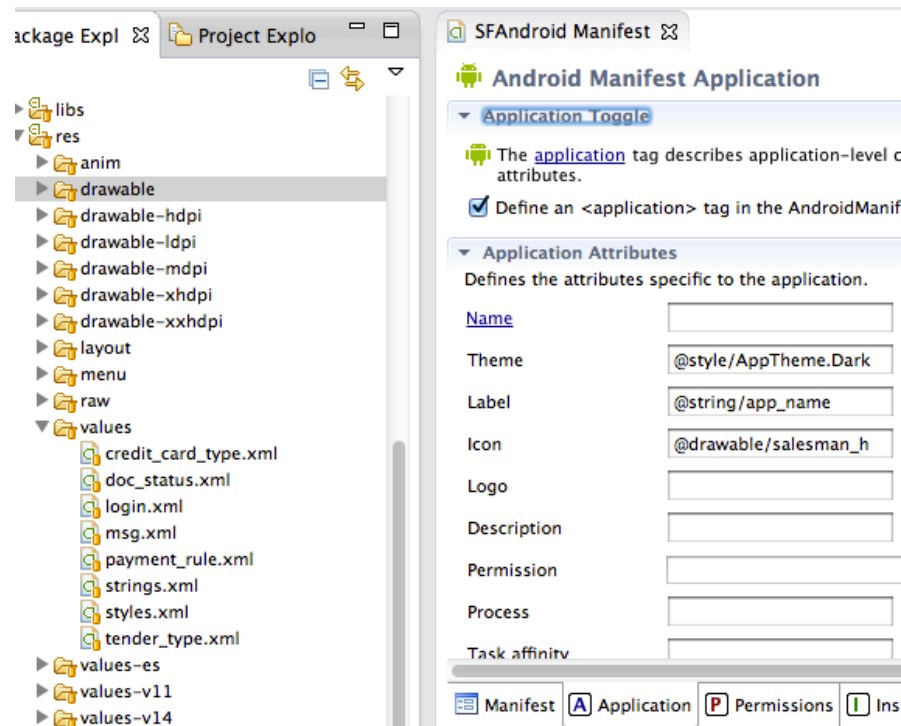
My source is at : <http://bitbucket.org/red1/sfandroid>. For newbies on Android programming you can refer to many resources online such as the official <http://developer.android.com/training/index.html> and another that is my favorite as it has good videos done in very comprehensive detail is at The New Boston site:

<http://thenewboston.org/list.php?cat=6>

After understanding how to setup the ADK (Android Development Kit) which is based on Eclipse IDE, you can checkout via its Mercurial add-on for the source code which is displayed here on the right.

The first thing we look at is the AndroidManifest.xml which is a standard in Android development. This is a big contribution from Google by the way. It is very popular, well supported and an amazing framework to learn. We can open up that manifest file and go to its first tab that is an UI for it as shown below.

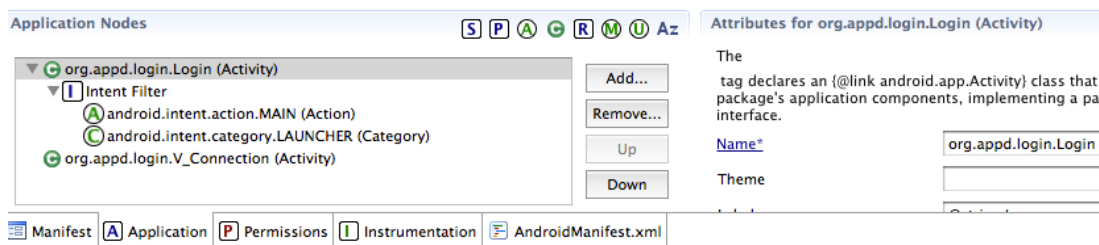




You can see the details and also other tabs at the bottom of the screen. Now at the **Application** tab we can see how it first choose to launch anything.

There is a **Theme** value which goes to [@style/AppTheme.Dark](#) and **Label** value which is [@string/app\\_name](#). There is also an **Icon** value that is [@drawable/salesman\\_h](#). These values are found in the folders within the **res** folder above at the *Package Explorer* view tab.

If you scroll the Manifest panel down lower, you can see an important part:



This part contains nodes that describe the ‘activity’ that the application will launch. Selecting the first activity, you will find on the right panel the code associated with it. Clicking on the *Name* link will pull out the code which in this case is *org.appd.login.Login*.

If you go to the rightmost tab you will see the manifest in the alternative native xml format shown below. We can inspect again the same settings.

```

<application
    android:allowBackup="true"
    android:icon="@drawable/salesman_h"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.Dark" >

    <activity
        android:name="org.appd.login.Login"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name="org.appd.login.V_Connection"
        android:label="@string/app_name" > </activity>
</application>

```

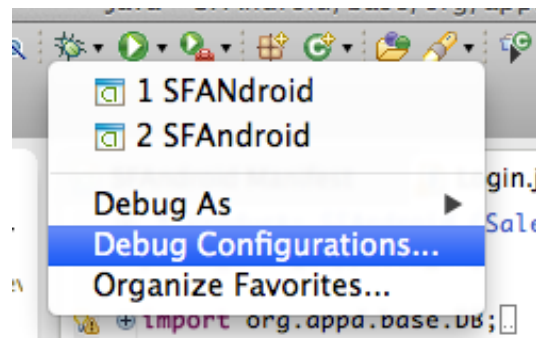
What is important is the `<application>` tag- stack shown above. The first part of the application stack defines the display icon for the welcome screen with its label and theme. Such artifacts are noted with `@<folder name>`/stored under the `/res/values` folder. In `/res/values/strings.xml` you will find the definition for the string `<string name="app_name">SF Android</string>`.

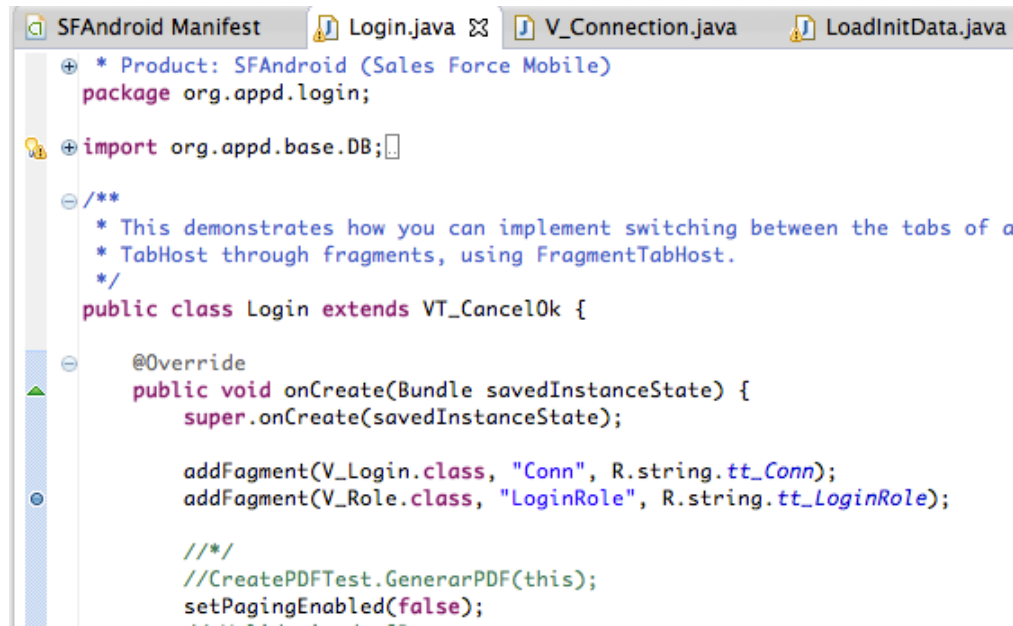
The `<activity>` tag defines the code to be launched, in the above mentions two classes: `"org.appd.login.Login"` and `"org.appd.login.V_Connection"`. The first has an `<intent-filter>` which matches that activity to the code that calls it via its intent method. It is a default intent and will run first. Next we see what happens in the `Login` class that it calls. We will also test it fully in a real Android device and step through the code.

## Launching via Emulator

You can also choose to launch the code via the in-built emulator instead of a real device.

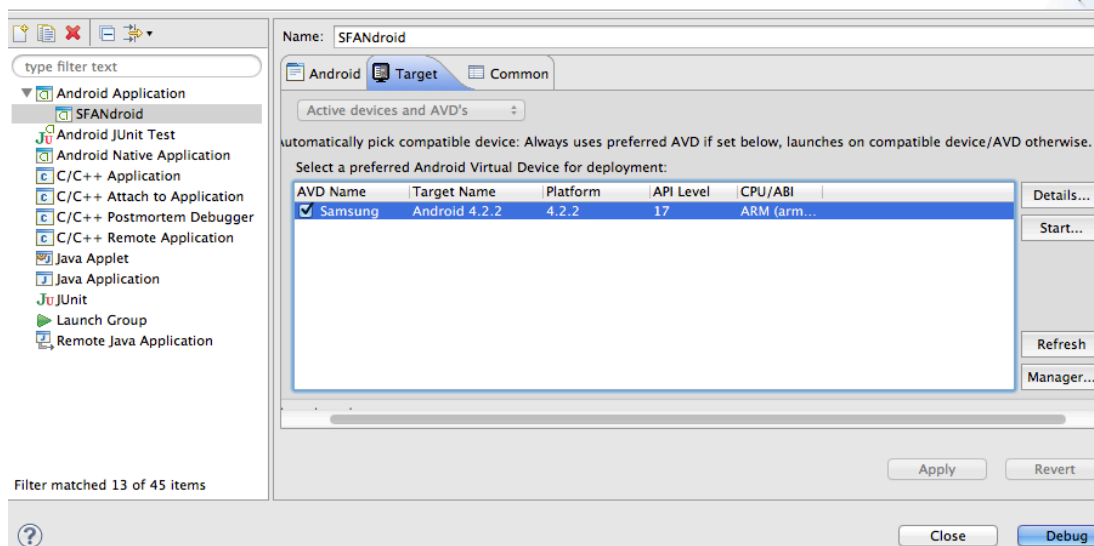
From the Debug icon in the ADK, choose Debug Configurations. Also put a debug break in the `Login` class somewhere so that we can confirm that everything works, i.e. the Debug, the emulator, the Manifest and the code. You can see that I have done so in the second screen below.





You can see that the first method that will run is the *onCreate(Bundle)* method because the extending of *VT\_CancelOK* is in turn extending *FragmentActivity* which does *onCreate(Bundle)*. So this overwrites that. Note also that *addFagment* is adding in two more tabs which will come out in the Android mobile screen when this code is running well. Back to the *Run Configurations*, below we add a *Samsung* emulator via the **Manager** at the bottom.

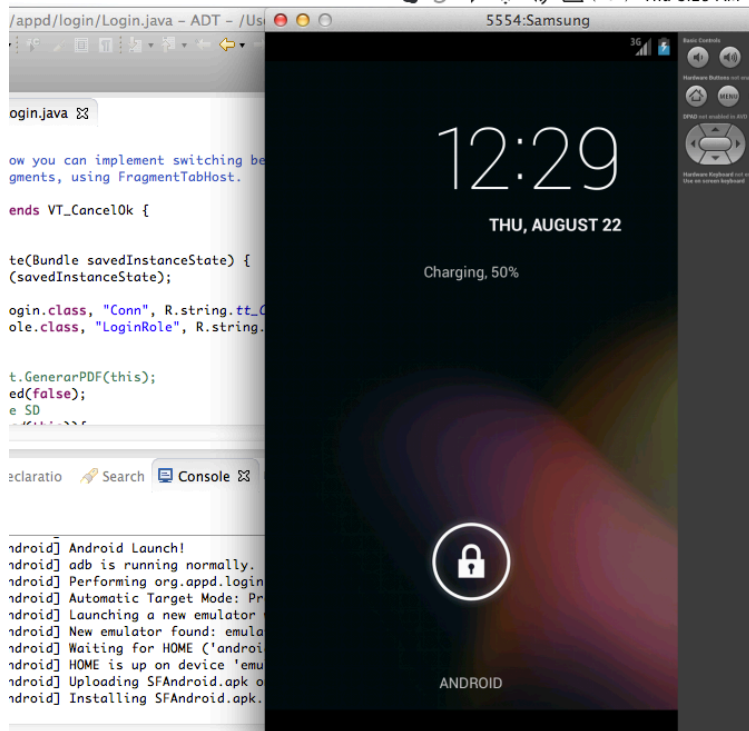
Create, manage, and run configurations  
Android Application



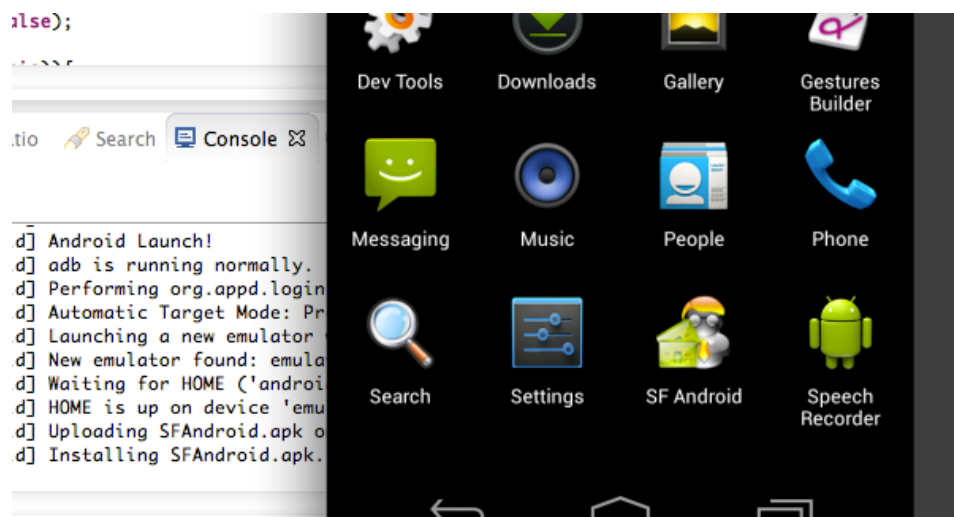
Once an emulator is added and checked, we can press the **Debug** button and see what happens.

You will see a pop-up *Samsung* emulated phone device on your desktop. If it is hidden just look up for it in the background.

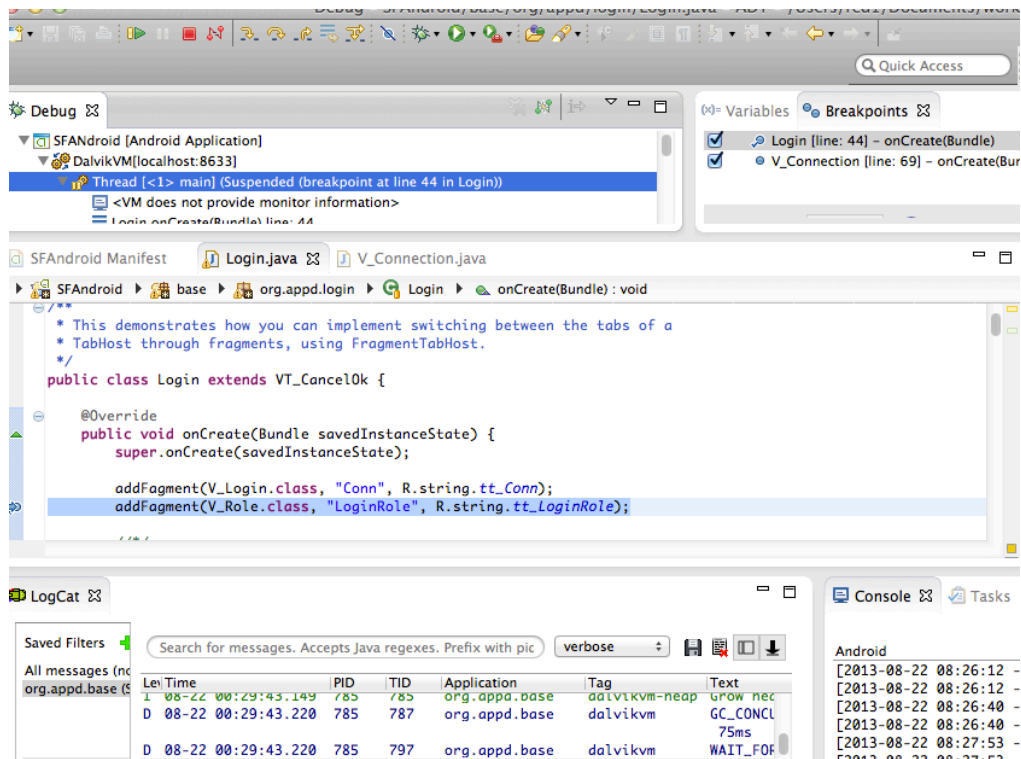
Switch it on by flicking the padlock icon to the side. Note also the Console showing prompts in its logs depicting the successful launch of Android and other processes performed before the SFAndroid project is uploaded to it. The last line `Installing SFAndroid.apk`, is



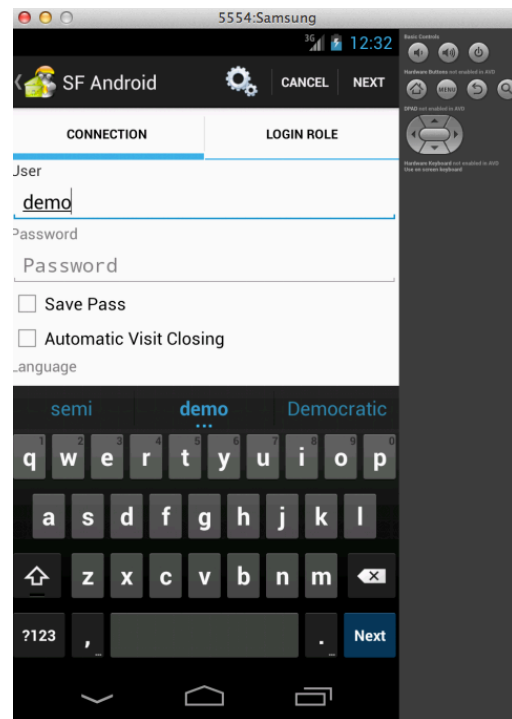
what you have to watch out for to confirm your set is working as expected. You then operate the 'mobile' phone as a normal Android phone and look for your apps. SFAndroid should be there as shown below. Double-tap on it and it should be running the code and hit the debug break.



## Login Code



The screen shows the break line stopping the execution of the code. As a Java developer familiar with Eclipse, you should know how to proceed by using the step-into, step-over buttons at the top of the Eclipse panel. At this juncture, we have proven that everything we wanted is working well. From this point onwards, the battle is nearly over. We have reached a complete proof-of-concept of the SFAndroid project that everything associated with it does work: server-side ERP setup and model, web-services, and now this mobile app. The final leg now will be the full cycle of this mobile app transmitting a request to the web-services to get into the ERP server and transmitting back a response to operate the ERP on the mobile phone. This is what we will expect onwards in this exercise. Right, you can see the first *Fragment* tab, **CONNECTION** in action. This emulation debugging allows you to examine each piece of code intact and in play.



At the Connection tab above in the Android phone, you are to give a User and Password. This must be present in the phone but it cannot be on initial launch. So entering anything will not mean anything but it is important to use the right credentials as during Initial Load this will be checked. So login with the usual [SuperUser/System](#).

In Android programming language this is done by a *Fragment* code called *V\_Login* as seen in the above screenshot of the *Fragment Activity Login* class and shown here below when highlighted.

```
addFagment(V_Login.class, "Conn", R.string.tt_Conn);
void org.appd.view.VT_CancelOk.addFagment(Class<?> clazz, String tag, int title)
Add Fagment Tab in FragmentActivity
```

It is adding a Tab which in Android programming language is referred to as a *Fragment Tab*. Its arguments requested are the class which is *V\_Login.java*, the *string tag* which is 'Conn' and the *title integer* is passed from the *res/values login.xml* entry for *tt\_Conn*:

```
<string name="tt_Conn">Connection</string>
```

Which gives the value displayed on the mobile screen tab title. You can try changing this value and see the result to understand how this works out. At this point you are beginning to get some grasp of the Android programming framework.

Returning to the code, at the **V\_Login** class.

```
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    et_User = (EditText) getActivity().findViewById(R.id.et_User);
    et_Pass = (EditText) getActivity().findViewById(R.id.et_Pass);
    ch_SavePass = (CheckBox) getActivity().findViewById(R.id.ch_SavePass);
    ch_AutomaticVisitClosing = (CheckBox) getActivity().findViewById(R.id.ch_AutomaticVisitClosing);
    sp_Language = (Cust_Spinner) getActivity().findViewById(R.id.sp_Language);
}
```

Here it is handling 5 items which the first two is important as the *acceptAction()* will *validUser()* by *findUser(user, pass)* from the *AD\_User* table.

```
V_Connection.java  V_Login.java  LoadInitData.java  Env.java
SfAndroid > base > org.appd.login > V_Connection > onCreate
@Override
@SuppressLint("NewApi")
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    super.setContentView(R.layout.v_connection);

    ActionBar actionBar = getActionBar();
    actionBar.setDisplayHomeAsUpEnabled(true);

    et_UrlSoap = (EditText) findViewById(R.id.et_UrlSoap);
    et_Method = (EditText) findViewById(R.id.et_Method);
}
```

After entering the credentials, we caught it at the next break in *V\_Connection* class.

The above is great news for the developer as s/he can easily work on the project with all the tools in perfect working condition. I could use the above ADK and Eclipse side-by-side both handling the server-side code of iDempiere and the WS in debug mode while the Android side running a phone emulator also in debug mode. The following screenshots will show the InitialLoad testing in action in such a manner. At the end I also made a movie out of it.

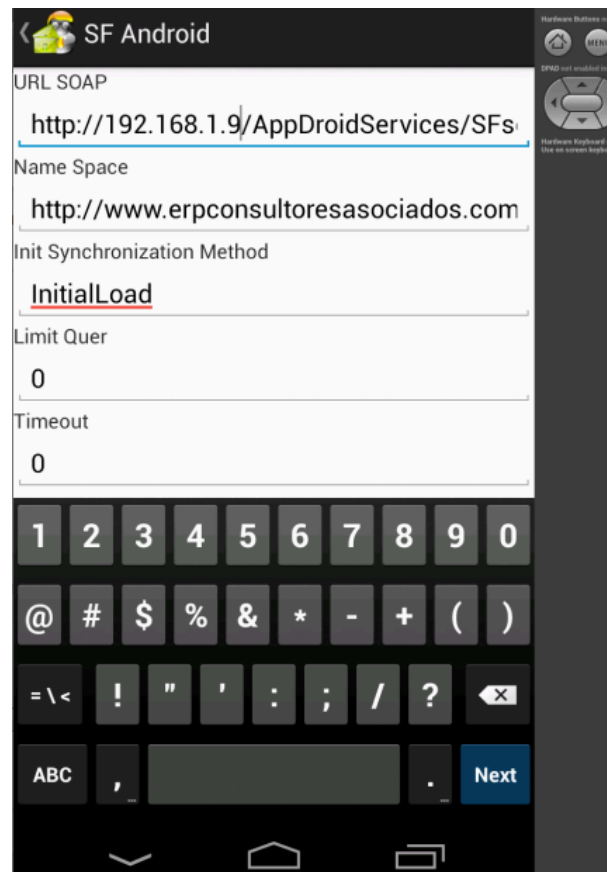
## SOAP Request

First the Android has to make a SOAP request via WS to the server-side. On the right shows the next screen after tapping on the **NEXT** button top right of the phone screen. It will display the **URL SOAP** field with a pre-set value. I have decode the field to allow edit on it. So now you can replace the 192.168.1.2 IP value with your own server IP. Note that you cannot use localhost if you are also running the server-side WS on your same computer. Find out what is your exact IP with a IPCONFIG command and use that. Remember also to put in the port of your WS. In my case its 8080 as shown on the next screen. Note that the emulator works like a real phone but it consumes alot of resources and so this can be slow on a lesser machine. Do not change the **Name Space**. You cannot do that anyway because the code does not allow it. If you want to allow a field to be editable you can look at how I did it for the URL SOAP field by peeking into this java class, *V\_Connection* - look at line 86 and 87:

```
//red1 allow user setting
et_UrlSoap.setText("http://192.168.1.2:8080/AppDroidServices/SFservices/AppDroidServices?wsdl");
et_UrlSoap.getText().toString().trim();
```

Put this after line 89 and you can make that field from read-only to editable:

```
et_NameSpace.getText().toString().trim();
```



Next you reduce the keypad to see the **Initial Synchronization** button for you to tap on. The `InitialLoad` class will activate and make a `SOAP.call()` via `HTTPTransport` API to the WS port. At first I have a hard time making this work due to a fault in the WS setting and Deepak Pansheriya helped me debug using the `TCPMON` tool which hacks the packets going to and fro between the ports. In the end he told me that the setting for WS in the interface class of `AppDroid-Service` is wrong. The right setting for its `SOAPBinding` is not `Style.RPC` but shown below:

The screenshot shows the 'SF Android' app interface. It has a dark header with a back arrow and the app name. Below the header, there are several input fields:
 

- URL SOAP**: `http://192.168.1.9:8080/AppDroidService`
- Name Space**: `http://www.erpconsultoresasociados.com`
- Init Synchronization Method**: `InitialLoad` (underlined in red)
- Limit Quer**: `0`
- Timeout**: `0`
- Save data SD(Slow)**: A checkbox that is checked.

 At the bottom, there is a large grey button labeled 'Initial Synchronization'.

```
@SOAPBinding(style=Style.DOCUMENT,use=Use.LITERAL,parameterStyle=ParameterStyle.WRAPPED)
```

To debug the **SOAP request - response** exchange, you can place breaks at the Android side in `InitialLoad` line 152

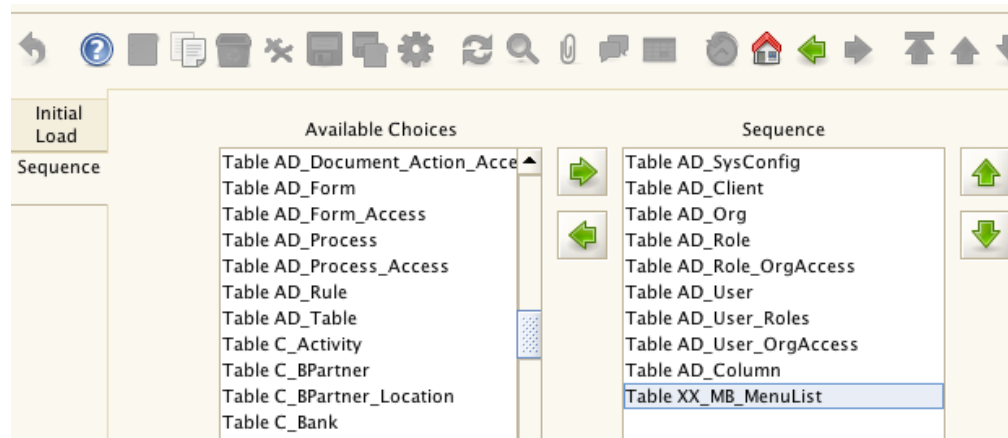
```
try {
    m_soap.call();
    return (SoapObject) m_soap.getM_Envelope().getResponse();
}
```

And to break at the server-side, `MAppDroidServicesImpl` line 61:

```
InitialLoadResponse rp = dataset.addNewInitialLoadResponse();
```

Of course you can use the **TCPMON** tool that Deepak taught me that can trouble-shoot packets for its exact payload. But the above code debugging is important for further improvement and extending of the project. It is also a good entry point for newbie developers who has no idea where to begin.

At the ERP side, I have selected only a certain number of records to send over as a test case as shown in the following screen. As you can see, the Initial Load window has a second tab to control easily the selection of records. All you have to do is just drag the items on the right panel towards the left and leave them there. You can also sort the items to instruct the Android synchronizing action on which item to load first into its `SQLite` database.

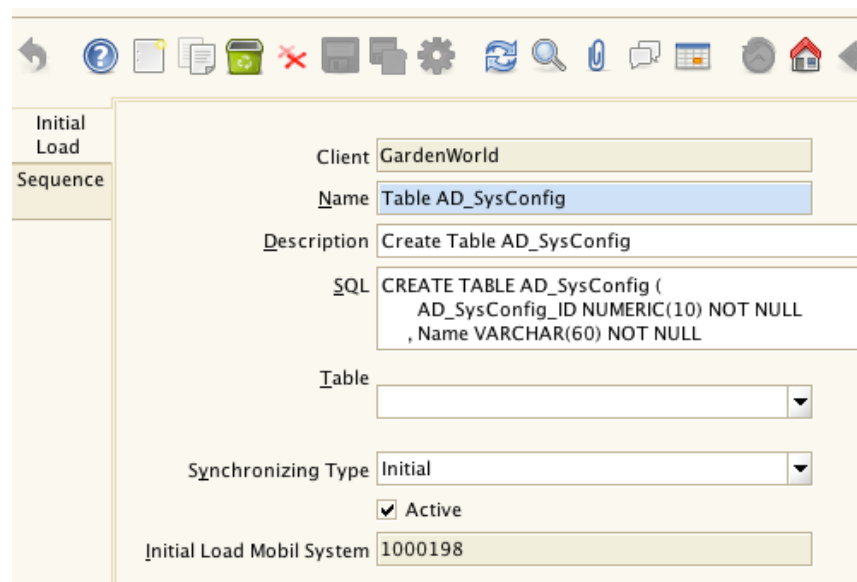


In this case, I made only 10 tables ready for action. The rest I drag them to the left panel. Those on the left are rendered `isActive = 'N'` and will not be used during synchronizing on the server's `MAppDroidServicesImpl` class due to line 51:

```
sql.append("select XXIL.*,TAB.TableName from XX_MB_InitialLoad XXIL Left Join
AD_Table TAB on XXIL.AD_Table_ID=TAB.AD_Table_ID Where XXIL.AD_Client_ID
='"+m_AD_Client_ID+"' And XXIL.isActive='Y' order by XXIL.SeqNo ");
```

In the snippet above, there is a check for `isActive` value.

The first tab (as a refresher from our earlier pages describing the server meta-data setup) below shows the content of SQL script that is meant to be executed into the Android side's SQLite database.



## SOAP Response

Now we turn our attention to the Android side which within a few milliseconds via the Internet will have received the SOAP response. The following two code snippets extract and execute the SQL content, highlighted in blue font:

```
public void initialLoad ()
{
    ArrayList<SORespInitialLoad> m_list = new ArrayList<SORespInitialLoad>();
    SoapObject m_soap = loadFromService();
    if(m_soap!=null)
    {
        m_list=convertSoapToArraylist(m_soap);
        if (loadInBD(m_list))
    }
}

-----
public ArrayList<SORespInitialLoad> convertSoapToArraylist(SoapObject m_soap)
{
    ArrayList<SORespInitialLoad> m_list = new ArrayList<SORespInitialLoad>();
    int m_tam = m_soap.getPropertyCount();
    for(int i=0;i<m_tam;i++)
    {
        SoapObject m_resp = (SoapObject)m_soap.getProperty(i);
        m_list.add(
            new SORespInitialLoad(
                m_resp.getPropertyAsString("name"), m_resp.getPropertyAsString("sql")));
    }
    return m_list;
}
```

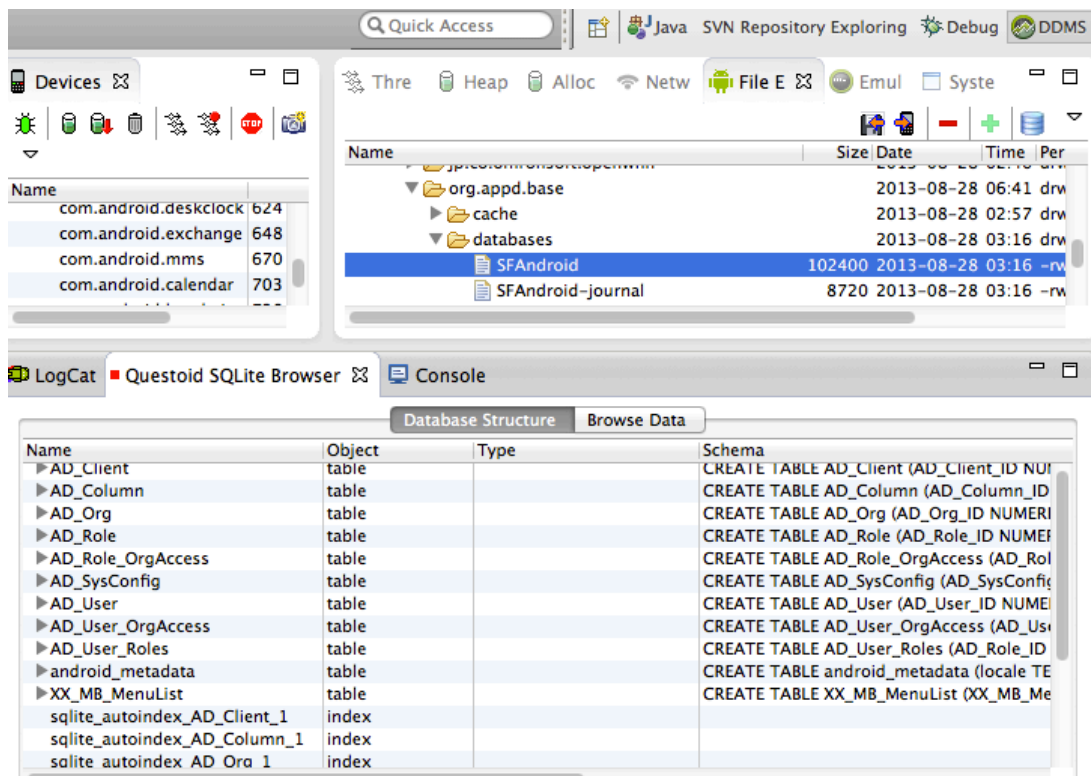
**August 31, Teluk Bahang, Penang.** When not traveling, not necessarily I end up nondescript. Here, Ai Ono, whom I met 2 months before in Tokyo, became really intrigued to come visit me for a few days and experience the village life, with its jovial and humid atmosphere.



## Using SQLite Browser

Now we can examine the final contents from the in-built database. From the ADK, we can use the DDMS plugin to examine. You can search for ‘[Questoid](#)’ or read about the plugin from this [forum](#).

To get to the SQLite Browser, select the DDMS perspective, go to the folder of `data/data/org.appd.base/databases/` and select `SFAndroid`, then click on the DB icon on the top right corner to select the view. The following screen is a guide.



You can see from the view panel, that all the SQLs we selected are committed into the database perfectly.

## Synchronizing Data

The next type of action in the *InitialLoad* window is the **Synchronizing** type. You can know this from the SQL script that is stored in the *InitialLoad* window as shown below:

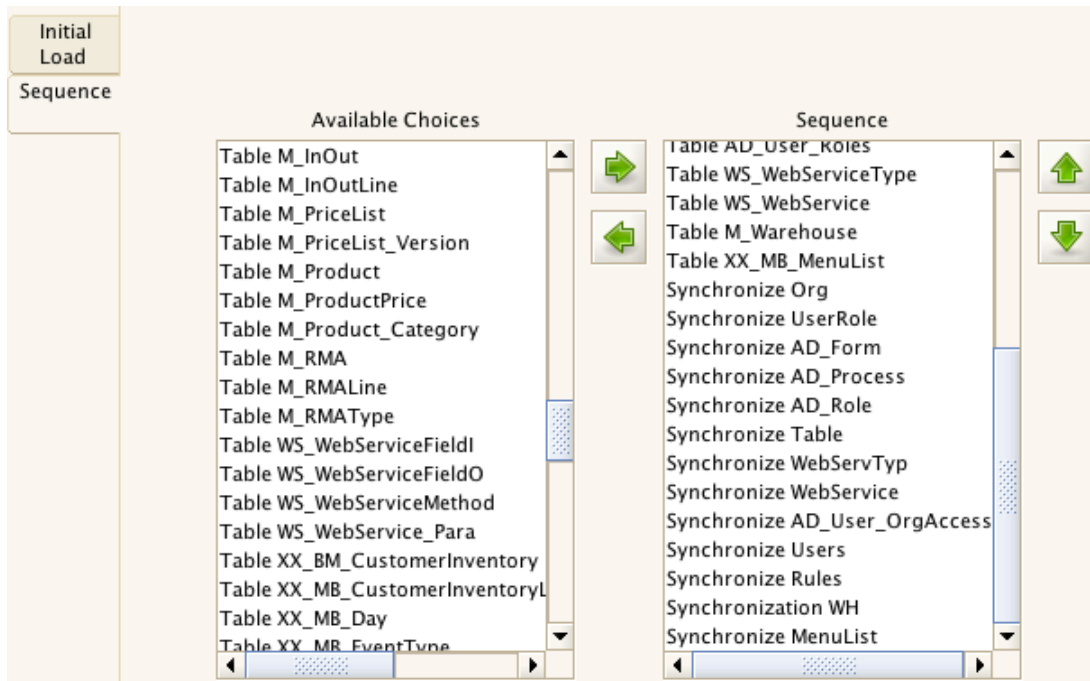
|              |                                                                                 |                                                 |
|--------------|---------------------------------------------------------------------------------|-------------------------------------------------|
| Initial Load | Client: GardenWorld                                                             | Organization: *                                 |
| Sequence     | Name: Synchronize Rules                                                         | Sequence: 100                                   |
|              | Description: Synchronize Rules                                                  |                                                 |
|              | SQL: INSERT INTO AD_Rule (AD_Rule_ID, Script) VALUES (\$AD_Rule_ID,\$Script\$); |                                                 |
|              | Table: AD_Rule_Rule                                                             | Sql WHERE: EntityType = 'APPD' AND IsActive='Y' |
|              | Synchronizing Type: Initial                                                     |                                                 |
|              | <input checked="" type="checkbox"/> Active                                      |                                                 |
|              | Initial Load Mobil System: 1000219                                              |                                                 |

It has an INSERT statement to get data from the server side into the Android side. The smart way to do it is the use of parsing markings '\$' which will extract the values from the *ResultSet*. Code on the server-side that does that is in the **MAppDroidServicesImpl** class, **LoadFromTable** method:

```
while(rsquery.next())
{
    l_sql = rs.getString("sql");
    while (l_sql.indexOf("$")>0)
    {
        l_init=l_sql.indexOf("$");
        l_end=l_sql.indexOf("$",l_init+1);
        l_campo = l_sql.substring(l_init+1,l_end);
        l_Value =transformValue(rsquery.getObject(l_campo));
        l_sql = l_sql.substring(0,l_init ) + l_Value+
            l_sql.substring(l_end+1,l_sql.length());
    }
    InitialLoadResponse rp = ds.addNewInitialLoadResponse();
    rp.setSql(l_sql);
    rp.setName(rs.getString("name"));
}
```

These **INSERT** content is then passed via the **InitialLoadResponse** in the **SOAP** array to the mobile side for execution into the SQLite database there.

Let's try it. On the next screen we drag some Synchronizing records into the InitialLoad, Sequence for transport. You can also do this right after the Tables transport earlier. They syncing action can be successive but those earlier successful items must be removed from the Sequence tab first. In the following screen it is not.



When the Mobile side Initial Synchronization button is clicked again, we can then examine the results in the internal database on the virtual emulator as seen in the [ADK's DDMS' Questoid SQLite Browser](#), at the *Browser Data* tab:

LogCat Questoid SQLite Browser Console

Database Structure Browse Data

Table: AD\_Role

| AD_Role_ID | Name                  | AD_Client_ID | S_AD_Role_ID | IsUseUserOrgAcce |
|------------|-----------------------|--------------|--------------|------------------|
| 103        | GardenWorld User      | 11           |              | N                |
| 50004      | Web Service Execution | 11           |              | N                |
| 0          | System Administrator  | 0            |              | N                |
| 102        | GardenWorld Admin     | 11           |              | N                |

You can see that the AD\_Role table data from the server-side has been successfully loaded into the mobile emulator.

## Trouble-shooting Synch Scripts

Since the scripts are hand-typed string content, typo mistakes can happen and here I show you how to solve the common mistakes. The first type happens when when we are synchronising a table that has not been **CREATE**d yet.

```
org.appd.base    SQL:      \-----, STORE_MENU , STORE_MENU ,11);
org.appd.base    SQL:      Insert Into AD_Org(AD_Org_ID,Name,Description,AD_
org.appd.base    SQL:      (12,'Store Central','GardenWorld Store!',11);
org.appd.base    SQL:      Insert Into AD_Org(AD_Org_ID,Name,Description,AD_
org.appd.base    SQLiteLog  (50007,'Stores',null,11);
org.appd.base    SQLiteLog  Insert Into AD_Process_Access(AD_Role_ID,AD_Proce
(1) no such table: AD_Process_Access
```

The above error in red is seen in the [LogCat](#) view panel. When you see this, then check your *Questoid* browser or remember back if you miss this in your InitialLoad Sequence tab during previous synch. Remedy is of course to **CREATE** the table first before the **INSERT**.

The following error stack shows a parsing exception at this line in the code, [convertSoapToArrayList](#):

| Tag            | Text                                                                                                                                  |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------|
| AndroidRuntime | FATAL EXCEPTION: AsyncTask #1                                                                                                         |
| AndroidRuntime | java.lang.RuntimeException: An error occured while executing doInBackground()                                                         |
| AndroidRuntime | at android.os.AsyncTask\$3.done(AsyncTask.java:299)                                                                                   |
| AndroidRuntime | at java.util.concurrent.FutureTask.finishCompletion(FutureTask.java:352)                                                              |
| AndroidRuntime | at java.util.concurrent.FutureTask.setException(FutureTask.java:219)                                                                  |
| AndroidRuntime | at java.util.concurrent.FutureTask.run(FutureTask.java:239)                                                                           |
| AndroidRuntime | at android.os.AsyncTask\$SerialExecutor\$1.run(AsyncTask.java:230)                                                                    |
| AndroidRuntime | at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1080)                                                    |
| AndroidRuntime | at java.util.concurrent.ThreadPoolExecutor\$Worker.run(ThreadPoolExecutor.java:573)                                                   |
| AndroidRuntime | at java.lang.Thread.run(Thread.java:856)                                                                                              |
| AndroidRuntime | Caused by: java.lang.ClassCastException: org.ksoap2.serialization.SoapPrimitive cannot be cast to org.ksoap2.serialization.SoapObject |
| AndroidRuntime | at org.appd.sync.read.InitialLoad.convertSoapToArrayList(InitialLoad.java:261)                                                        |
| AndroidRuntime | at org.appd.sync.read.InitialLoad.initialLoad(InitialLoad.java:184)                                                                   |

This is due to a missing '\$' tag in the SQL script. The remedy is to put back the tag and run the synch action again from the mobile app.

|             |                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------|
| Client      | GardenWorld                                                                                                  |
| Name        | Synchronization                                                                                              |
| Description | Sincronizaci#n XX_MB_MenuList                                                                                |
| SQL         | (\$XX_MB_MenuList_ID\$, \$AD_Form_ID \$OrderByClause\$, \$XX_MB_ParentMenuType\$, Name\$, \$Description\$, : |
| Table       | XX_MB_MenuList_Menu of Mobile Syst                                                                           |

## Using ADB command line tools

There was once i could not get to see the contents of the data in the DDMS Questoid viewer. I tried another tool that can. It is the ADB tool under [AndroidSDK/sdk/platform-tools](#). Below is the screen-shot of it in action.

```
Redhuans-MacBook-Pro-2:platform-tools red1$ ./adb -s emulator-5554 shell
root@android:/ # sqlite3 /data/data/org.appd.base/databases/SFAndroid
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> SELECT * FROM XX_MB_MENULIST
...> ;
1000017|Y|download_h|Updated Between '@#ReadPayTerms_LastSyncDate@' And now() |||1000017|
1000026|Y|download_h|Updated Between '@#ReadPriceListVersions_LastSyncDate@' And now() |||1000026|
1000027|Y|download_h|Updated Between '@#ReadUOMs_LastSyncDate@' And now() |||1000018|N|
1000030|Y|download_h|Updated Between '@#ReadProducts_LastSyncDate@' And now() |||1000018|
1000029|Y|download_h|Updated Between '@#ReadProductPrices_LastSyncDate@' And now() |||1000029|
1000021|Y|download_h|Updated Between '@#ReadBPartners_LastSyncDate@' And now() |||1000021|
1000055|Y|download_h|Updated Between '@#ReadC_City_LastSyncDate@' And now() |||1000020|
1000055|Y|download_h|Updated Between '@#ReadC_City_LastSyncDate@' And now() |||1000020|
```

First you CD to the platform-tools directory in your development space via a terminal window. Then execute the ADB shell with the command:

```
./adb -s emulator-5554 shell
```

The port number 5554 is known with the command, `./adb devices`.

Then call SQLite3 editor pointing to the location of your database:

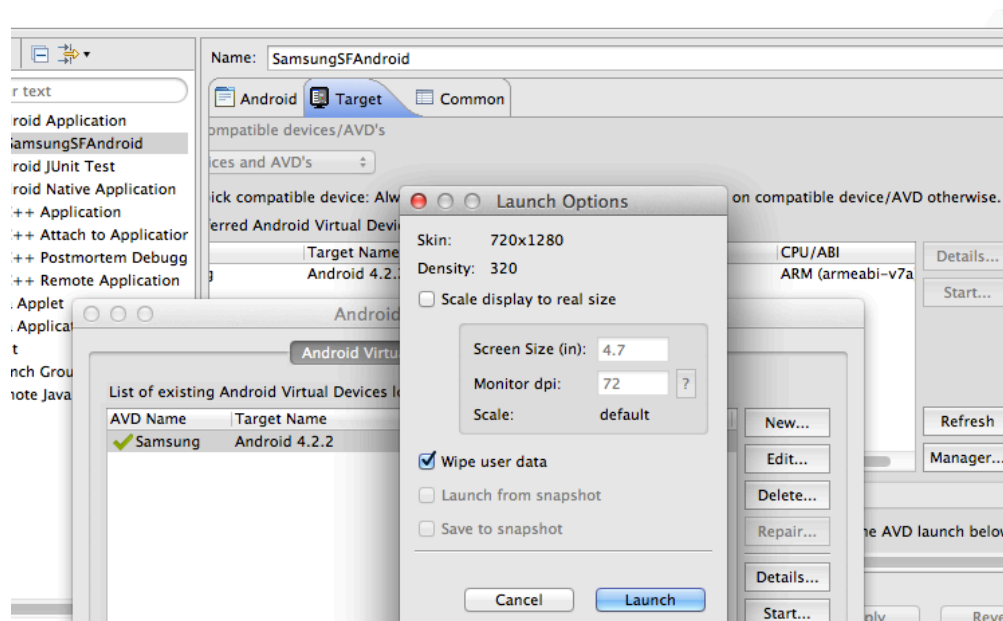
```
sqlite3 /data/data/org.appd.base/databases/SFAndroid
```

At the sqlite3 prompt you can begin issuing SQL scripts and commands ending with semi-colons as shown in the screen capture above.

## Wipe-out User Data

As always in testing, you can hit some error in data handling or code that you need to scratch the database back to the beginning. In the case of Android's DB, it starts from nothing, or not having any user data of the folder structure as you see in the last screen. You can wipe out any created data by specifying in the launch configuration at the check-box doing so. Then during launch a pop-up dialog will ask you if you wish to do so or leave the data alone. That is convenient as you can decide at every launch.

In case you want to wipe-out on a blank launch and not during the SFAndroid launch you can follow the series of screens described below.



First at the launch configuration for **Run** or **Debug**, select the **Target** tab.

Then scroll down right section of the panel to see a **Manager** button and click it.

Choose your *Android Virtual Device* and look for the **Start** button and click on it for a **Launch Options** box pop-up. Check the **Wipe user data** check-box and then press **Launch**.

## EMBEDDED DEMO

The Android mobile app has a ready-made demo done by the Venezuelan team and we shall see how it is done so that usual developers can also do the same. The demo also allows to test the SFAndroid further as sample data interacts with the activity.xmls and mobile code. It is invaluable resource to learn from and for many developers, referring to live examples is the best way to learn.

(The synchronising that you see on the previous section happens when you try it out in a simulator's internal memory and not on SD card in your ADK. Somehow the code does not manage to load the embedded database when it detects an empty internal memory, where `Env.IsLoad(ctx)` is false. This allows an *InitialLoad* connection to the WS. By right there should be an option to initialize even with a loaded Env. in case we need to reset. We shall leave this feature inquiry for the future. As of now, we already conveniently arrived at the intended purpose.)

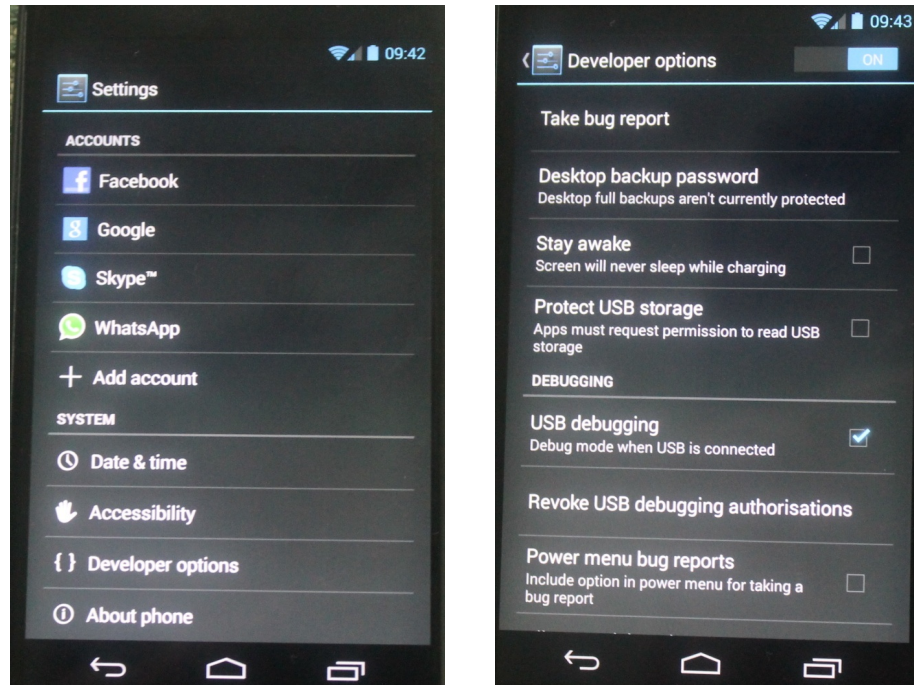
The embedded demo sample is loaded without synching via WS yet. In a way it is an embedded database seed during installation. It also serves as a good strategy during development and debugging to reuse the idea and formulate new samples or starter packs for standard end-users.

**August 29, Templer's Park Waterfalls.** Ai Ono with Hadeesa from Birmingham and Farida from Paris, both volunteers at a local orphanage near my farm home. Together with my wife (in pink) it is a rare chance for myself to appreciate my own country.



## Debug ADK on real Phone

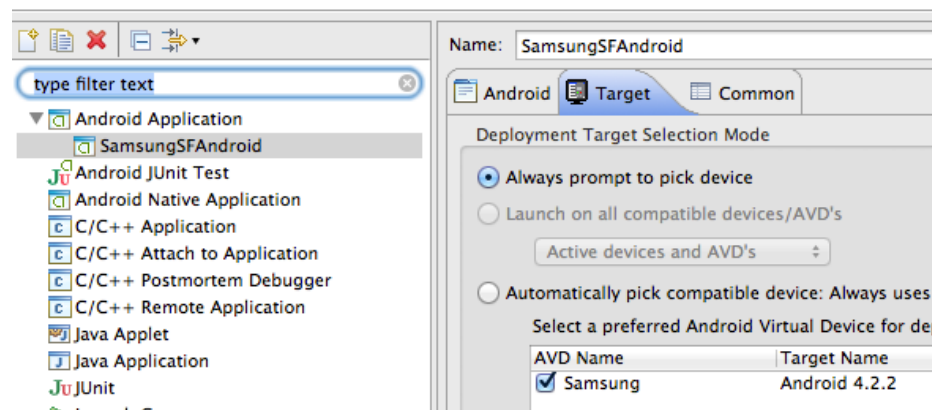
One real-life reason why you will prefer to test on a real phone is the speed it works is many times faster than on an emulator. Firstly, we need to connect to our actual phone device via USB cable to your computer with the ADK running. You have to enable the Developer Options from the Settings as shown in the following screen-shots.



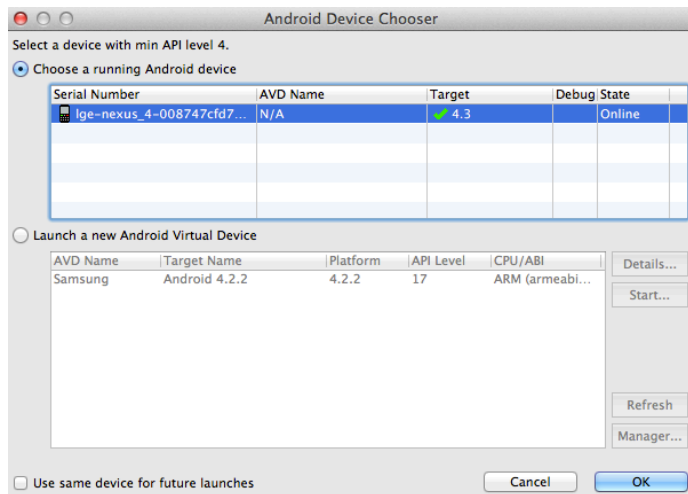
We are deploying the SFAndroid to the phone in hacker style. Later versions we shall learn to deploy it as a *Google Play Store* app. Ensure that in your ADK's Debug Configurations, is set to pick up the target device:

### Create, manage, and run configurations

Android Application

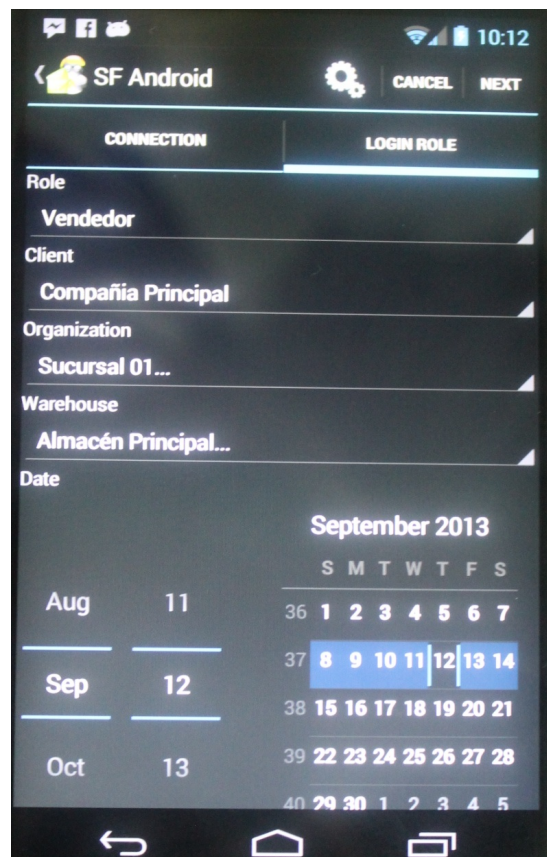


After clicking on the Debug or Run As button in your ADK, you can see the device appearing on the panel. If it does not appear, reconnect the USB cable. If it still does not appear, do not ask me. Ask Google. It helps me in times like these. In fact all the time. In my case the detection and selection of the device works remarkably trouble-free.



Once the app started running your phone should be on ready mode. Unlock the padlock to make it ready. The SFAndroid screen shall appear. The login/pass should be 'Demo/Demo'. Pressing on the *Next* button will lead to the *Login Role* screen:

The Login Role screen has the values populated from the embedded database seed. These samples are from the Venezuelan team and thus they are in Spanish. Next, we take a look at how this embedded database is obtained during the Initial Load process.



## Source of Embedded Sample

The embedded sample can be found under the folder: [res/raw/sfandroid](#). It is just an SQLite database that was in use and copied over into the resource folder to be picked up during the initial loading process. Now this only happens if the **SD card** is in use either at the emulator or in the phone.

The code that picks up this seed is in the LoadInitData.java at Line 79:

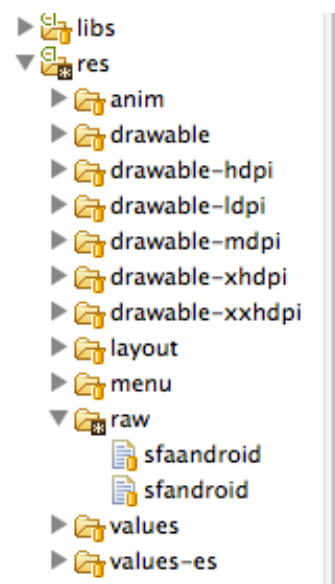
```
InputStream is =
ctx.getResources().openRawResource(R.raw.sfandroid);
```

The above `R.raw.sfandroid` snip says that the database is located at 'res/raw' folder of the project. Once you get to the above screen on your phone that means the SQLite database is created and exists under the path:

```
String dbpath = Environment.getExternalStorageDirectory() +
DB.getDB_Path();
```

whose value is [storage/emulated/0/ERP/data/SFAndroid](#). This means the data is stored at that location as an SQLite database format.

This is a good idea for adaptation to any new distro where sample or setup data is already pre-loaded for easier use. Such utilities from Android environment is a sure killer for today's exploding mobile market.

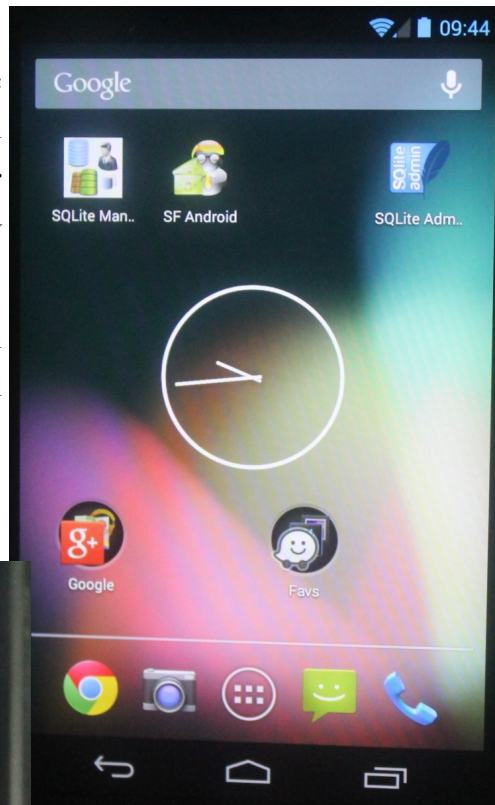
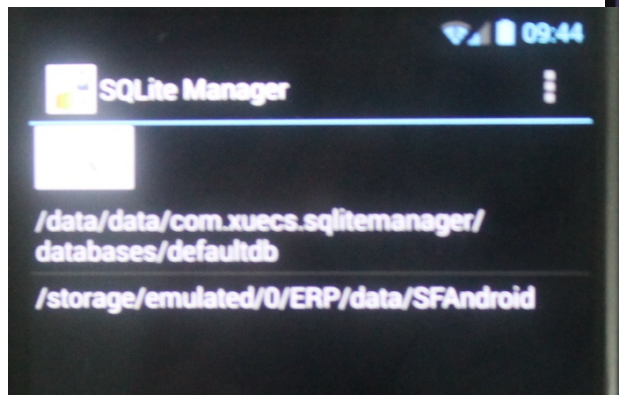


## SQLite Data Admin

Well, why not we really find that out inside the phone itself? It will use another tool that you can download from the *Google Play Store*. Look for an SQLite database manager or editor. In my case I got it here in my Android:

Once you downloaded it for install you can open the **SQLite Manager** and note the path stated earlier.

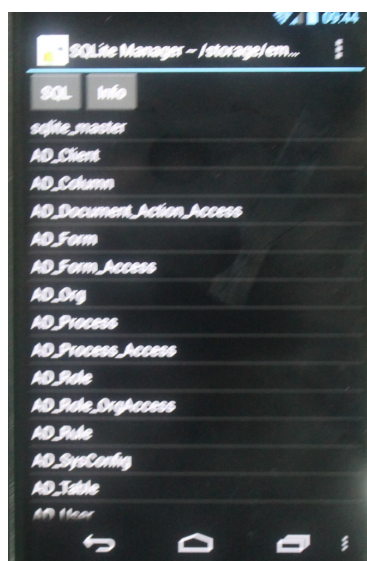
Tap on the lower line to get inside that location.



You will then see the tables already existing inside SFAndroid database.

You can then tap into any of them to view their records. Note the SQL

button on the top left of the tables. You can run SQL scripts such as clear the tables or search for any records according to a certain filter criteria.

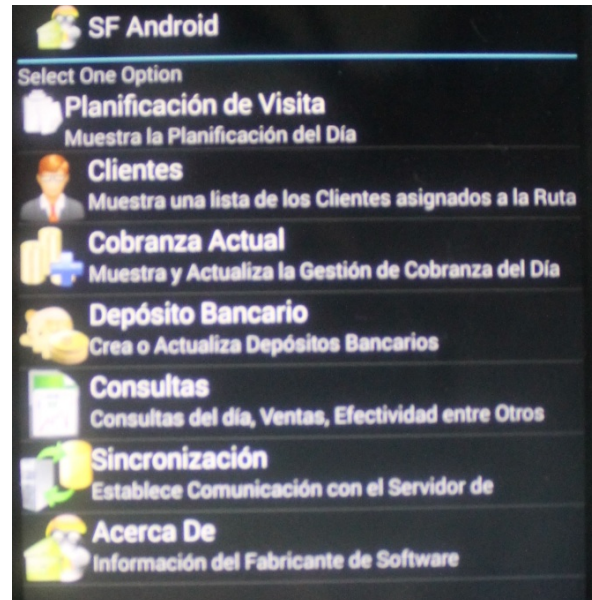


| C_BPPartner_ID | Value  | TaxID  | Name                   | Name2 | So_CreditLimit | So_Cr... |
|----------------|--------|--------|------------------------|-------|----------------|----------|
| 1003240        | 213463 | 213463 | BIANCA .               | ***** | 0              | 0        |
| 1000973        | PE1155 | PE1155 | DIS BIAO               | ***** | 5000           | 11395    |
| 1002047        | 199408 | 199408 | VENTAS AGRICOLAS PENAS | ***** | 0              | 6615     |
| 1002051        | 477293 | 477293 | DIS MERCADO DE ABASTOS | ***** | 0              | 0        |
| 1001370        | 287261 | 287261 | MINIMARKET XTRA        | ***** | 450000         | 306794   |
| 1002192        | N18388 | N18388 | DIS LOS AMIGOS         | ***** | 0              | 0        |
| 1001903        | 200976 | 200976 | CONCEPCION HIDALGO     | ***** | 3000           | 9323.5   |
| 1002674        | 412213 | 412213 | DIS FELIX              | ***** | 0              | 0        |
| 1000980        | 108787 | 108787 | VENTAS GRAN AEROPUERTO | ***** | 3500           | 2390.4E  |
| 1002673        | E89212 | E89212 | DEPÓSITO EL GATO       | ***** | 0              | 0        |
| 1002050        | 830419 | 830419 | - SANDRA               | ***** | 3000           | 2094     |
| 1001897        | N20411 | N20411 | MAX LA CASA            | ***** | 0              | 0        |
| 1001977        | 308152 | 308152 | MR CARNES              | ***** | 25000          | 48254.E  |
| 1001898        | 883315 | 883315 | MINIMARKET LA GENEROSA | ***** | 0              | 0        |

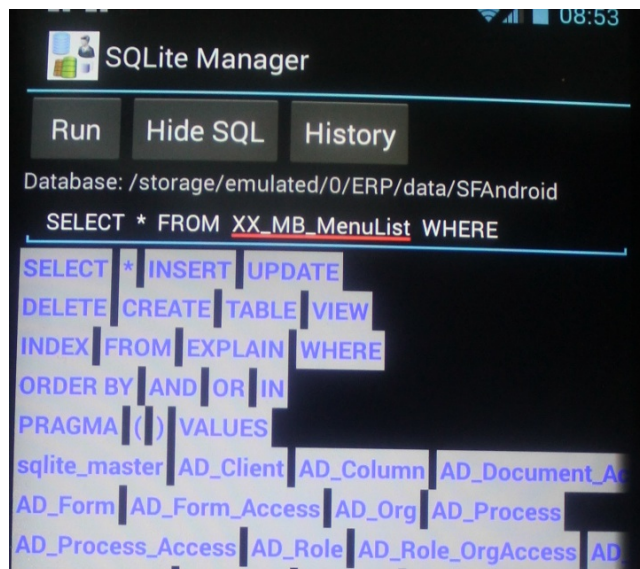
## SFAndroid Meta Menu

You probably know by the now the concept of Application Dictionary or AD and the story of using meta-data to define an application structure. As I asserted early on, this SFAndroid is a marvel that indeed reuses such a concept in its application.

Since we can now get the demo sample application running on the phone, we shall examine under the hood how its menu appears ready to execute its underlying menu item's code class.



It is quite simple in essence. It is the `XX_MB_MenuList` data calling `AD_Form` data. Data upon data, it is the metadata idea. We can clearly see inside these screenshots. First we go into the [SQLite3 Manager](#) to view the first table and note the `Ad_Form_ID`. Since not all menu items are program calls we use an SQL statement to filter for those. We can tap on the [SQL](#) editor and tap on the ready made



snips to form `'SELECT * FROM XX_MB_MenuList WHERE AD_Form_ID > 0'`. This is to fetch MenuList records that have `AD_Form_ID` integer values i.e. pointing to the IDs in the `AD_Form` table.

After tapping on [RUN](#), scroll to the bottom of the phone to see the results.

| XX_MB_MenuList_ID | AD_Form_ID | IsActive | Action | ImgName         | AD_Table_ID | Whe                                   |
|-------------------|------------|----------|--------|-----------------|-------------|---------------------------------------|
| 1000009           | 1000027    | Y        | F      | syncserver_h    | N           | 130                                   |
| 1000064           | 1000028    | Y        | F      | salesman_h      | N           | 140                                   |
| 1000100           | 1000032    | Y        | F      | collect_money_h | 407         | DATE(C_Cash.State<br>DATE('@#DateP@') |
| 1000063           | 1000020    | Y        | F      | klipper_h       | 1000078     | XX_MB_PlanningVis                     |

There are 13 such records. Then we go to the [AD\\_Form](#) table and view the corresponding records. We can see that their [ClassName](#) column contains values that correspond to actual Java classes in the SFAndroid project.

| AD_Form_ID | Name                   | ClassName                          | IsActive | Descriptio  |
|------------|------------------------|------------------------------------|----------|-------------|
| 1000023    | MB Clientes            | org.appd.view.MT_BPartner          | Y        | Consulta l  |
| 1000027    | MB Sincronización      | org.appd.view.MV_Synchronizing     | Y        | MB Sincro   |
| 1000030    | MB RMA                 | org.appd.view.MT_CustomerRMA       | Y        | Consulta c  |
| 1000020    | MB Menu Planning Visit | org.appd.view.M_MenuPlanningVisit  | Y        | Menú Mos    |
| 1000031    | MB Visit               | org.appd.view.MT_VisitAndLine      | Y        | Visit       |
| 1000021    | MB Pedidos             | org.appd.view.MT_CustomerOrder     | Y        | Consulta c  |
| 1000028    | MB About               | org.appd.view.FT_About             | Y        | Info Client |
| 1000029    | MB Inventory           | org.appd.view.MT_CustomerInventory | Y        | Toma de l   |
| 1000033    | MB Deposit             | org.appd.view.MT_Deposit           | Y        | Deposit     |
| 1000032    | MB Collection          | org.appd.view.MT_Cash              | Y        | Collection  |

So this is how it works. The [MV\\_Menu](#) class traverse through the [MenuList](#) table and where there are Forms, fetches its respective class strings for ClassLoading when that item is selected. You can compare these classnames with the SFAndroid menu in the previous page. You can examine the [org.appd.menu.view.MV\\_Menu.java](#) in the source to see how it is done, particularly at [load\(\)](#) and [loadActivity\(DisplayMenuItem item\)](#).

(to be cont'd) Future versions will focus on actual end-user use or more deeper into the SFAndroid Venezuelan application for Salesforce mobile management.

### Your chance to contribute

At this juncture, it is a huge step into the future of this project for iDempiere and the community behind it. Nothing can stop any developer or user to think about all the things that can be done. With the source code readily freedom-ready for improvement, we can stand on the shoulders of giants and move faster ahead.

The development team in Venezuela is still ongoing with its new version where we can watch their repository for ongoing changes. Also you can watch mine as well. Further versions of this project will be updated accordingly in those locations and this document too.

So step onto it and be part of a global team and interesting things await all of us.

**May 27, Stuttgart** - With Heinrette Baumann, co-author of UML In Action. She and her husband invited me to stopover during my trip back from Slovakia to Dusseldorf to discuss about 'the war'. It was a delight to know that they are writers themselves having produced that UML book 10 years ago in the German language before it got translated by **Packt Books UK** into the English medium. I requested them for an autographed copy.

